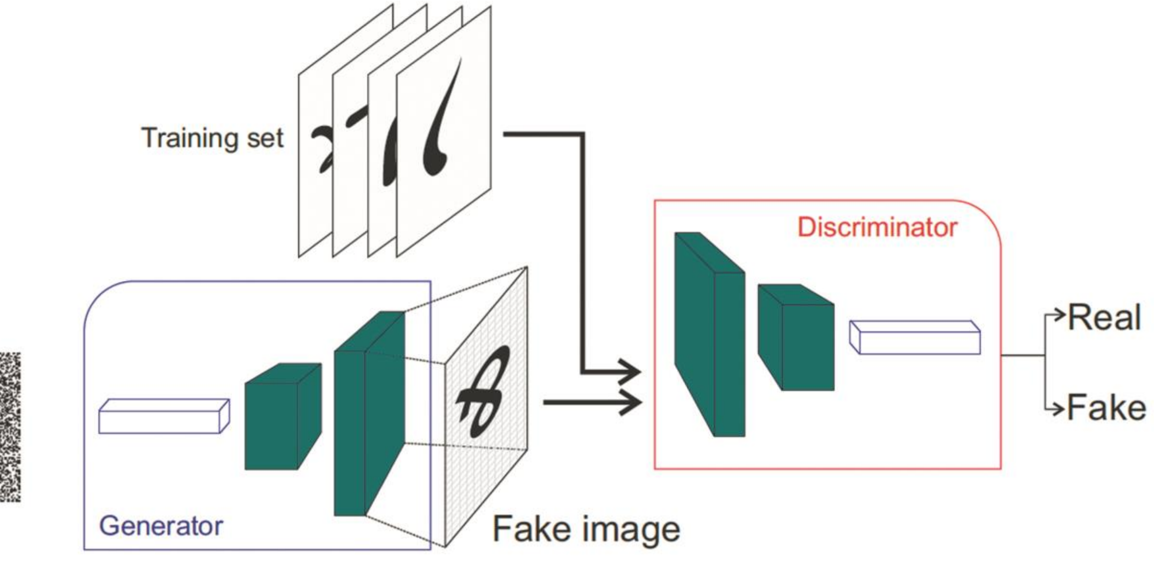
شبكات الخصومة التوليدية

نماذج تعلم عميق تم حلها باستخدام شبكات الخصومة التوليدية GANs

ترجمة واعداد: د. علاء طعيمة

GANs





بســمه تعالى

شبكات الخصومة التوليدية

مشاريع تعلم عميق تم حلها باستخدام شبكات الخصومة التوليدية GANs

ترجمة واعداد:

د. علاء طعيمة

مقدمة المترجم

تُعد شبكات الخصومة التوليدية Generative Adversarial Networks (GANs) واحدة من أكثر المواضيع إثارة للاهتمامفي التعلم الآلي اليوم. لقد تم استخدامهافي عدد من المشكلات (وليس فقط لإنشاء أرقام MNIST!) وكان أداؤها جيدًا للغايةفي كل حالة. تتكون شبكة GAN من مولد generator ومميز discriminator، يتنافسان ضد بعضهما البعض لتحقيق نتائج مذهلة.

تعتمد شبكات GAN على سيناريو يشبه اللعبة حيث يلعب المولد والمميز ضد بعضهما البعض. يحاول المولد إنشاء بيانات تشبه البيانات الحقيقية، بينما يهدف المميز إلى التمييز بين البيانات الحقيقية والمزيفة.

تم تطبيق شبكات GAN على العديد من التطبيقات، بمافي ذلك توليد الصور والتنبؤ بالفيديو وإنشاء الكائنات ثلاثية الأبعاد.، تحل شبكات GAN العديد من المشكلات وتخلق فرصًا جديدةفي صناعات متعددة.في هذ الكتاب، سوف نستكشف ما هي شبكات GAN ,ومعماريتها، ودوال الخطأ، وتطبيقاتها، وتنفيذ العديد من المشاريع باستخدام شبكات GAN.

لقد حاولت قدر المستطاع ان اترجم المقالات والمشاريع الأكثر طرحاًفي مجال شبكات الخصومة التوليدية GANs مع الشرح المناسب والكافي، ومع هذا يبقى عملاً بشرياً يحتمل النقص، فاذا كان لديك أي ملاحظات حول هذا الكتاب، فلا تتردد بمراسلتنا عبر بريدنا الالكتروني alaa.taima@qu.edu.iq .

نأمل ان يساعد هذا الكتاب كل من يريد ان يدخلفي مجال التعلم العميق وشبكات الخصومة التوليدية ومساعدة القارئ العربي على تعلم هذا المجال. اسأل الله التوفيقفي هذا العمل لأثراء المحتوى العربي الذي يفتقر أشد الافتقار إلى محتوى جيد ورصينفي مجال التعلم الآلي والتعلم العميق وعلم البيانات. ونرجو لك الاستمتاع مع الكتاب ولا تنسونا من صالح الدعاء.

د. علاء طعيمة

كلية علوم الحاسوب وتكنولوجيا المعلومات

جامعة القادسية / العراق

المحتويات

0) مقدمة شاملة لشبكات الخصومة التوليدية (GANs) An End-to-End Introduction to Generative Adversarial Networks(GANs)

هناك العديد من الطرق التي يمكن من خلالها تعليم الآلة كيفية إنشاء مخرجات على البيانات غير المرئية. لقد ترك التقدم التكنولوجيفي مختلف القطاعات الجميعفي حالة صدمة. نحن الآنفي مرحلة أصبح فيها التعلم العميق deep learning والشبكات العصبية neural networks قوية جداً بحيث يمكنها إنشاء وجه بشري جديد من الصفر لم يكن موجودًا من قبل ولكنه يبدو حقيقيًا بناءً على بعض البيانات المدربة. هذه التقنية ليست سوى GAN (شبكة الخصومة التوليدية Generative Adversarial Network) وهي موضوع دراستنا.

ما هي شبكات الخصومة التوليدية (GANs)؟

تم تطوير شبكات الخصومة التوليدية (GANs)في عام 2014 على يد إيان جودفيلو Ian Goodfellow وزملائه. تعد GANفي الأساس طريقة للنمذجة التوليدية generative modeling التي تولد مجموعة جديدة من البيانات بناءً على بيانات التدريب التي تشبه بيانات التدريب. تحتوي شبكات GAN على كتلتين رئيسيتين (شبكتين عصبيتين) تتنافسان مع بعضهما البعض وتكونان قادرين على التقاط ونسخ وتحليل الاختلافاتفي مجموعة البيانات. يُطلق على النموذجين عادةً اسم المولد Generator والمميز Discriminator، وسنغطيهمافي المكونات الموجودة على شبكات GAN. لفهم مصطلح GAN، دعونا نقسمه إلى ثلاثة أجزاء منفصلة:

- **التوليدي Generative:** لتعلم نموذج توليدي، يصف كيفية إنشاء البيانات من حيث النموذج الاحتمالي. بكلمات بسيطة، يشرح كيفية إنشاء البيانات بشكل مرئي.
- **الخصومة Adversarial**: يتم تدريب النموذجفي بيئة عدائية.
- **الشبكات Networks:** استخدم الشبكات العصبية العميقة لأغراض التدريب.

تأخذ شبكة المولد generator network مدخلات عشوائية (ضوضاء عادة noise) وتقوم بإنشاء عينات samples، مثل الصور أو النصوص أو الصوت، التي تشبه بيانات التدريب التي تم تدريبها عليها. الهدف من المولد هو إنتاج عينات لا يمكن تمييزها عن البيانات الحقيقية.

ومن ناحية أخرى، تحاول شبكة المميز discriminator network التمييز بين العينات الحقيقية والمولدة. يتم تدريبه باستخدام عينات حقيقية من بيانات التدريب والعينات التي تم إنشاؤها من المولد. هدف المميز هو تصنيف البيانات الحقيقية بشكل صحيح على أنها حقيقية والبيانات الناتجة على أنها مزيفة.

تتضمن عملية التدريب لعبة عدائية adversarial game بين المولد والمميز. يهدف المولد إلى إنتاج عينات تخدع المُميِّز، بينما يحاول المُميِّز تحسين قدرته على التمييز بين البيانات الحقيقية والمولدة. يدفع هذا التدريب العدائي كلا الشبكتين إلى التحسن مع مرور الوقت.

مع تقدم التدريب، يصبح المولد أكثر مهارةفي إنتاج عينات واقعية، بينما يصبح المُميِّز أكثر مهارةفي التمييز بين البيانات الحقيقية والمولدة. ومن الناحية المثالية، تتقارب هذه العملية إلى نقطة يكون فيها المولد قادرًا على توليد عينات عالية الجودة يصعب على المُميِّز تمييزها عن البيانات الحقيقية.

لقد أظهرت شبكات GAN نتائج مبهرةفي مجالات مختلفة، مثل تركيب الصور image synthesis، وإنشاء النص text generation، وحتى إنشاء الفيديو video generation. لقد تم استخدامها لمهام مثل إنشاء صور واقعية، وإنشاء صور مزيفة، وتحسين الصور منخفضة الدقة، والمزيد. لقد طورت شبكات GAN بشكل كبير مجال النمذجة التوليدية وفتحت إمكانيات جديدة للتطبيقات الإبداعيةفي الذكاء الاصطناعي.

لماذا تم تطوير شبكات GAN؟

يمكن بسهولة خداع خوارزميات التعلم الآلي والشبكات العصبية لإساءة تصنيف الأشياء عن طريق إضافة قدر من الضوضاء إلى البيانات. بعد إضافة قدر من الضوضاء، تزداد فرص التصنيف الخاطئ للصور. ومن هنا كان الارتفاع الطفيفـفي إمكانية تنفيذ شيء يمكن للشبكات العصبية أن تبدأفي تصور أنماط جديدة مثل عينات بيانات التدريب. وهكذا تم إنشاء شبكات GAN التي تولد نتائج مزيفة جديدة مشابهة للنتائج الأصلية.

تطبيقات شبكات الخصومة التوليدية (GANs)

إن القراءة عن شبكات GAN أمر مثير للغاية، وعندما تقرأ تطبيقها، آمل أن تصل الإثارة إلى عنان السماء ومن ثم فإن دراسة عمل شبكات GAN تخلق تأثيرًا مختلفًا على التعلم.

1. توليد بيانات جديدة من البيانات المتاحة Generate new data from available data: ويعني توليد عينات جديدة من عينة متاحة لا تشبه العينة الحقيقية.
2. إنشاء صور واقعية لأشخاص لم يكونوا موجودين من قبل.
3. لا يقتصر GANs على الصور، بل يمكنه إنشاء نصوص ومقالات وأغاني وقصائد وما إلى ذلك.
4. إنشاء موسيقى باستخدام بعض الأصوات المستنسخة Generate Music by using some clone Voice: إذا قدمت بعض الأصوات، فيمكن لشبكات GAN إنشاء ميزة استنساخ مماثلة لها.في هذه الورقة البحثية، اقترح باحثون من معهد NITفي طوكيو نظامًا

قادرًا على توليد ألحان من كلمات الأغاني بمساعدة العلاقات المكتسبة بين النوتات الموسيقية والموضوعات.

5. إنشاء النص إلى صورة Text to Image Generation (Object GAN و Object Driven GAN)
6. إنشاء شخصيات أنمي في تطوير الألعاب وإنتاج الرسوم المتحركة.
7. ترجمة الصور إلى صور Image to Image Translation: يمكننا ترجمة صورة إلى أخرى دون تغيير خلفية الصورة المصدر. على سبيل المثال، يمكن لـ GANs استبدال كلب بقطة.
8. دقة منخفضة إلى دقة عالية Low resolution to High resolution: إذا قمت بتمرير صورة أو مقطع فيديو منخفض الدقة، فيمكن لـ GAN إنتاج نسخة صورة عالية الدقة من نفس الشيء.
9. التنبؤ بالإطار التالي في الفيديو Prediction of Next Frame in the video: من خلال تدريب الشبكة العصبية على إطارات صغيرة من الفيديو، تستطيع شبكات GAN إنشاء أو التنبؤ بإطار صغير تالي من الفيديو.
10. إنشاء الصور التفاعلية Interactive Image Generation: يعني ذلك أن شبكات GAN قادرة على إنشاء صور ولقطات فيديوفي شكل فني إذا تم تدريبها على مجموعة البيانات الحقيقية الصحيحة.
11. الكلام Speech: نشر باحثون من كلية لندن مؤخرًا نظامًا يسمى GAN-TTS والذي يتعلم كيفية إنشاء صوت خام من خلال التدريب على 567 مجموعة من بيانات الكلام.

إن الأبحاث حول شبكات GAN تصل إلى ذروتها، وفي السنوات القادمة سنرى شبكات GAN تنتج فيديو وصوت وصورًا عالية الجودة. وكما هو الحال بالفعل، تعاونت Microsoft مع OpenAI للعمل على GPT واستكشاف قوة GAN في المستوى التالي.

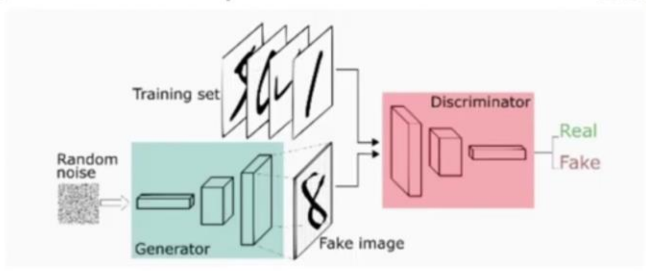
مكونات شبكات الخصومة التوليدية (GANs)

ما هو الحدس الهندسي وراء عمل شبكات GAN؟

المكونان الرئيسيان لشبكات GAN هما المولد Generator والمميز Discriminator. إن دور المولد يشبه اللص في إنشاء عينات مزيفة بناءً على العينة الأصلية وجعل التمييز أحمقًا لفهم أنها مزيفة على أنها حقيقية. من ناحية أخرى، فإن المُميِّز يشبه الشرطة التي يتمثل دورها في تحديد العيوب في العينات التي أنشأها المولد وتصنيفها على أنها مزيفة أو حقيقية. تستمر هذه المنافسة بين كلا المكونين حتى يتم الوصول إلى مستوى الكمال حيث يفوز المولد مما يجعل المميز يُخدع بالبيانات المزيفة.



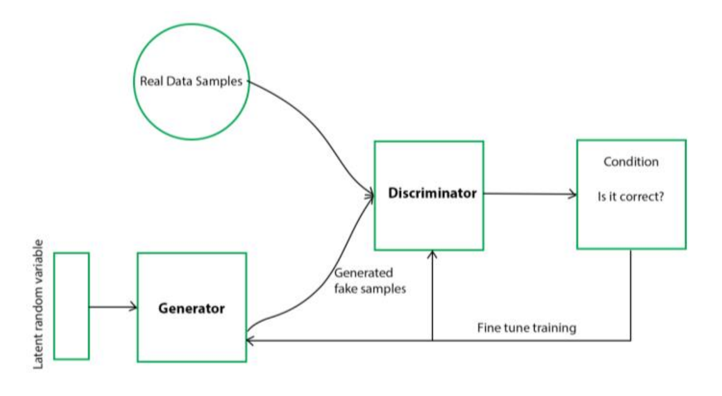




الآن دعونا نفهم ما هو هذا المكون الثنائي لفهم عملية تدريب GAN بشكل حدسي.

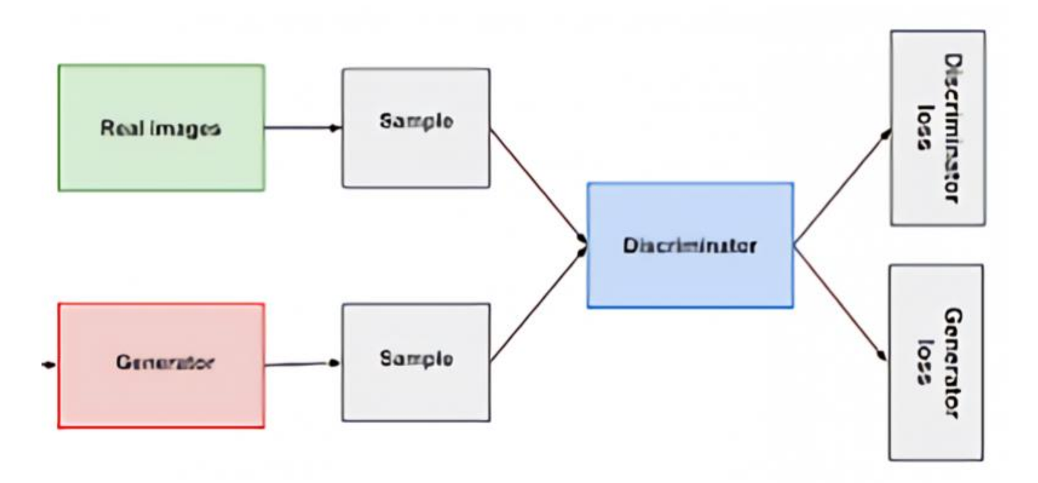
1) المميز Discriminator: وهو أسلوب التعلم خاضع للإشراف supervised learning approach يعني أنه مصنف بسيط يتنبأ بأن البيانات مزيفة أو حقيقية. يتم تدريبه على بيانات حقيقية ويقدم تعليقات للمولد.

2) المولد Generator: وهو اسلوب التعلم غير الخاضع للأشراف unsupervised learning approach. سيتم إنشاء بيانات مزيفة بناءً على البيانات الأصلية (الحقيقية). وهي أيضًا شبكة عصبية تحتوي على طبقات مخفية hidden layers ودالة التنشيط activation والخسارة (الخطأ) loss function. هدفها هو توليد الصورة المزيفة بناءً على ردود الفعل وجعل المُميز يخدع أنه لا يستطيع التنبؤ بالصورة المزيفة. وعندما يخدع المولد المُميِّز، يتوقف التدريب ويمكننا القول إنه تم إنشاء نموذج GAN المعمم.



هنا يلتقط النموذج التوليدي generative model توزيع البيانات ويتم تدريبه بهذه الطريقة لتوليد العينة الجديدة التي تحاول تعظيم احتمالية ارتكاب المُميِّز لخطأ (تعظيم خطأ المُميِّز maximize discriminator loss). من ناحية أخرى، يعتمد التمييز على نموذج يقدر احتمالية أن تكون العينة التي يتلقاها من بيانات التدريب وليس من المولد ويحاول تصنيفها بدقة وتقليل دقة GAN. ومن ثم تم صياغة شبكة GAN كلعبة minimax حيث يحاول المميز تقليل مكافأته V(D, G) ويحاول المولد زيادة خطأ المميز إلى الحد الأقصى.

ربما تتساءل الآن كيف يتم إنشاء المعمارية الفعلية لشبكة GAN، وكيف يتم بناء شبكتين عصبيتين ويتم التدريب والتنبؤ بهما؟ لتبسيط الأمر، قم بإلقاء نظرة على المعمارية العامة لـ GAN أدناه.



نحن نعلم أن كلا المكونين عبارة عن شبكات عصبية. يمكننا أن نرى أن خرج المولد متصل مباشرة بإدخال المُميِّز. والمميز يتنبأ به ومن خلال الانتشار الخلفي backpropagation يستقبل المولد إشارة تغذية راجعة لتحديث الأوزان وتحسين الأداء. المُميِّز عبارة عن شبكة عصبية ذات امامية التغذية feed-forward neural network.

التدريب والتنبؤ بشبكات الخصومة التوليدية (GANs)

نحن نعرف الحدس الهندسي geometric intuition لـ GAN، والآن دعونا نفهم تدريب GAN. في هذا القسم، سيكون تدريب المولد والمميز واضحًا لك بشكل منفصل.

الخطوة 1) تحديد المشكلة

بيان المشكلة هو مفتاح نجاح المشروع لذا فإن الخطوة الأولى هي تحديد مشكلتك. تعمل شبكات GAN مع مجموعة مختلفة من المشكلات التي تستهدفها، لذا تحتاج إلى تحديد ما تقوم بإنشائه مثل الصوت والقصيدة والنص والصورة هي نوع من المشاكل.

الخطوة 2) تحديد معمارية GAN

هناك العديد من أنواع GAN المختلفة، والتي سندرسها بمزيد من التفصيل. يتعين علينا تحديد نوع معمارية GAN التي نستخدمها.

الخطوة 3) تدريب المميز على مجموعة البيانات الحقيقية

الآن يتم تدريب المميز على مجموعة بيانات حقيقية. إنه يحتوي فقط على مسار للأمام، ولا يوجد انتشار خلفيفي تدريب المُميِّزفي n من الفترات epochs. والبيانات التي تقدمها هي بدون ضوضاء وتحتوي فقط على صور حقيقية، وبالنسبة للصور المزيفة، يستخدم المميز المثيلات instances التي أنشأها المولد كمخرجات سلبية. الآن، ماذا يحدثفي وقت التدريب على التمييز.

ويصنف كلا من البيانات الحقيقية والمزيفة.

يساعد خطأ المميز discriminator loss على تحسين أدائها ومعاقبتها عندما تصنف بشكل خاطئ على أنها حقيقية أو مزيفة أو العكس.

يتم تحديث أوزان المُميز من خلال خطأ المُميز.

الخطوة 4) تدريب المولدات

قم بتوفير بعض المدخلات المزيفة للمولد (الضوضاء) وسيستخدم بعض الضوضاء العشوائية random noise ويولد بعض المخرجات المزيفة. عندما يتم تدريب المولد، يكون المميز خاملاً، وعندما يتم تدريب المميز، يكون المولد خاملاً. أثناء تدريب المولد على أي ضوضاء عشوائية كمدخل، فإنه يحاول تحويله إلى بيانات ذات معنى. يستغرق الحصول على مخرجات ذات معنى من المولد وقتًا ويستمرفي العديد من العصور. خطوات تدريب المولد مذكورة أدناه.

- الحصول على ضوضاء عشوائية وإنتاج مخرجات المولد على عينة الضوضاء.
- توقع إخراج المولد من التمييز على أنه أصلي أو مزيف.
- نحسب خطأ المميز.
- إجراء الانتشار الخلفي من خلال المميز والمولد لحساب التدرجات gradients.
- استخدم التدرجات لتحديث أوزان المولد.

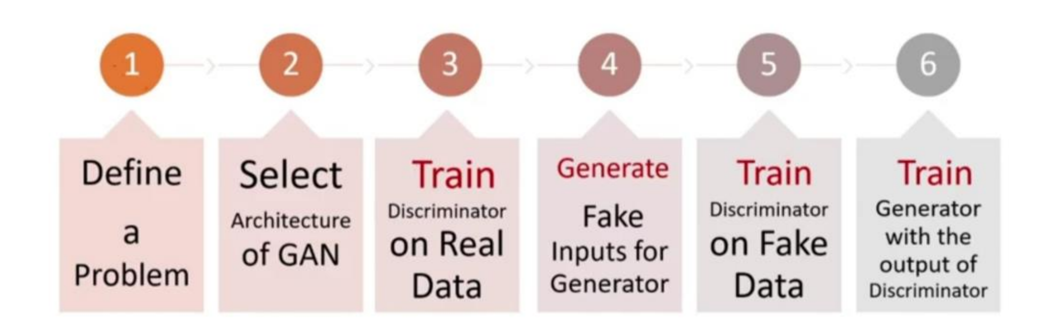
الخطوة 5) تدريب المميز على البيانات المزيفة

سيتم تمرير العينات التي تم إنشاؤها بواسطة المولد إلى المميز وسوف يتنبأ بأن البيانات التي تم تمريرها إليها مزيفة أو حقيقية وستقدم تعليقات إلى المولد مرة أخرى.

الخطوة 6) تدريب المولد بمخرجات المميز

مرة أخرى، سيتم تدريب المولد على التعليقات المقدمة من المميز ومحاولة تحسين الأداء.

هذه عملية متكررة وتستمرفي العمل حتى لا ينجح المولدفي خداع المميز.



دالة خطأ شبكات الخصومة التوليدية (GANs).

آمل أن يكون عمل شبكة GAN مفهومًا تمامًا، والآن دعونا نفهم دالة الخسارة (الخطأ) loss function التي تستخدمها وتقليلها وتعظيمهافي هذه العملية التكرارية. يحاول المولد تقليل دالة الخطأ التالية بينما يحاول المُميِّز تعظيمها. إنها نفس لعبة Minimax إذاكنت قد لعبت من قبل.

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))]$$

- D(x) هو تقدير المُميِّز لاحتمال أن يكون مثيل البيانات الحقيقي x حقيقيًا.
- Ex هي القيمة المتوقعة على كافة مثيلات البيانات الحقيقية.
- G(z) هو خرج المولد عند إعطاء الضوضاء z.
- D(G(z)) هو تقدير المُميِّز لاحتمال أن يكون المثيل المزيف حقيقيًا.
- Ez هي القيمة المتوقعة على جميع المدخلات العشوائية للمولد (في الواقع، القيمة المتوقعة على جميع المثيلات المزيفة التي تم إنشاؤها (G(z)).

- الصيغة مشتقة من الانتروبيا المتقاطعة cross-entropy.

التحديات التي تواجهها شبكات الخصومة التوليدية (GANs)

1) **مشكلة الثبات بين المولد والمميز:** لا نريد أن يكون هذا المميز صارمًا للغاية، بل نريد أن يكون متساهلا.
2) **مشكلة في تحديد موضع الكائنات:** لنفترض أن لدينافي الصورة 3 أحصنة وأن المولد قد أنشأ 6 عيون وحصانًا واحدًا.
3) **المشكلة في فهم الأشياء العالمية:** لا تفهم شبكات GAN المعمارية العالمية أو المعمارية الشاملة التي تشبه مشكلة المنظور. وهذا يعني أنهفي بعض الأحيان تقوم GAN بإنشاء صورة غير واقعية ولا يمكن أن تكون ممكنة.
4) **مشكلة في فهم المنظور:** لا يمكنه فهم الصور ثلاثية الأبعاد وإذا قمنا بتدريبه على مثل هذه الأنواع من الصور فسوف يفشلفي إنشاء صور ثلاثية الأبعاد لأن شبكات GAN اليوم قادرة على العمل على صور أحادية الأبعاد.

أنواع مختلفة من شبكات الخصومة التوليدية (GANs)

1) **DC GAN**: إنها شبكة GAN تلافيفية عميقة (Deep convolutional GAN). إنه أحد أكثر أنواع معمارية GAN استخدامًا وقوة ونجاحًا. يتم تنفيذه بمساعدة ConvNets بدلاً من بيرسيبترون متعدد الطبقات Multi-layered perceptron. تستخدم شبكات ConvNets خطوة تلافيفية convolutional stride ويتم إنشاؤها بدون تجميع الحد الأقصى max pooling ولا تكون الطبقاتفي هذه الشبكة متصلة بشكل كامل.
2) **GAN المشروطة وGAN غير المشروطة (CGAN)**: GAN المشروطة (Conditional GAN) هي شبكة عصبية للتعلم العميق يتم فيها استخدام بعض المعلمات الإضافية. يتم أيضًا وضع التسميات Labelsفي مدخلات المميز لمساعدة المُميز على تصنيف المدخلات بشكل صحيح وعدم امتلاءها بسهولة بواسطة المولد.
3) **GAN للمربع الأصغر (LSGAN)Least Square GAN**: هو نوع من GAN يتبنى دالة خطأ المربع الأصغر least-square loss function للمميز. يؤدي تقليل دالة الهدف لـ LSGAN إلى تقليل انحراف بيرسون Pearson divergence.
4) **المصنف المساعد GAN (ACGAN)Auxilary Classifier GAN**: وهو نفس CGAN ونسخة متقدمة منه. تنص على أن المميز لا ينبغي أن تصنف الصورة على أنها حقيقية أو مزيفة فحسب، بل يجب أيضًا أن توفر تسمية المصدر أو الفئة لصورة الإدخال.

5) **Dual Video Discriminator GAN – DVD-GAN:** عبارة عن شبكة خصومة توليدية لتوليد الفيديو مبنية على معمارية BigGAN. يستخدم DVD-GAN مُميزين: المُميز المكاني Spatial Discriminator والمُميز الزمني Temporal Discriminator.

6) **SRGAN**: وظيفته الرئيسية هي تحويل الدقة المنخفضة إلى دقة عالية المعروفة باسم تحويل المجال Domain Transformation.

7) **Cycle GAN**: تم إصداره في عام 2017 والذي يقوم بمهمة ترجمة الصور Image Translation. لنفترض أننا قمنا بتدريبها على مجموعة بيانات صور الخيول ويمكننا ترجمتها إلى صور حمار وحشي.

8) **Info GAN**: إصدار متقدم من GAN قادر على تعلم كيفية فك تشابك التمثيل disentangle representation في نهج التعلم غير الخاضع للإشراف.

خطوات تنفيذ GAN الأساسية

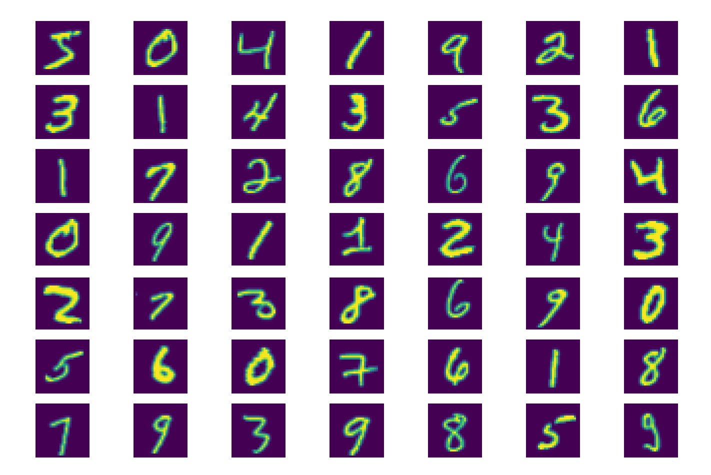
- استيراد كافة المكتبات.
- الحصول على مجموعة البيانات.
- إعداد البيانات: يتضمن خطوات مختلفة لإنجازها مثل المعالجة المسبقة للبيانات، وتوسيع نطاقها، وتسويتها، وإعادة تشكيلها.
- تحديد دالة المولد والمميز.
- انشاء ضوضاء عشوائية ثم إنشاء صورة بضوضاء عشوائية.
- إعداد المعلمات مثل تحديد الفترة وحجم الدفعة وحجم العينة.
- تحديد دالة توليد الصور كعينات.
- تدريب المميز بعد ذلك تدريب المولد وسيقوم بإنشاء الصور.
- سوف نرى مدى وضوح الصور الذي تم إنشاؤه بواسطة المولد.

التنفيذ العملي لـ(GANs) على مجموعة بيانات MNIST

سنتبع الآن جميع الخطوات المذكورة أعلاه من خلال تطبيق GAN على مجموعة بيانات شائعة جداً تُعرف باسم مجموعة بيانات MNIST.

حول مجموعة البيانات

مجموعة بيانات MNIST هي مجموعة بيانات شائعة جداً لصور الأرقام المكتوبة بخط اليد بين 0 إلى 9 في شكل تدرج رمادي بحجم 28*28. ويوجد إجمالي 60000 صورة لهذه الصور المربعة الصغيرة في مجموعة بيانات MNIST.



هدفنا هو تدريب نموذج المميز باستخدام مجموعة بيانات MNIST ومع بعض الضوضاء وبعد تقديم بعض ضوضاء العينة مثل نموذج MNIST إلى المولد لإنشاء نفس المعلومات مثل مجموعة بيانات MNIST التي تعطي صورًا دقيقة أو أصلية ولكن تم إنشاؤها بالفعل بواسطة نموذج المولد. فلنبدأ باستيراد المكتبات التي نحتاجها.

استيراد المكتبات

تعد مكتبات الاستيراد مفيدة للمعالجة المسبقة والتحويل وإنشاء نموذج الشبكة العصبية.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, LeakyReLU,
Dropout, BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam
```

تحميل مجموعة بيانات MNIST

عندما نقوم بتحميل مجموعة البيانات المضمنة من أي مكتبة، ففي معظم الأحيان يتم تقسيمها بالفعل إلى مجموعة تدريب واختبار train and test set، لذلك سنقوم بتحميل مجموعة البيانات إلى نموذجين مختلفين.

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Scale the inputs in range of (-1, +1) for better training
x_train, x_test = x_train / 255.0 * 2 - 1, x_test / 255.0 *
2 - 1
```

إذا كنت تريد رسم بعض الأمثلة على الصور من مجموعة البيانات، فيمكنك ببساطة رسمها من مجموعة بيانات التدريب باستخدام matplotlib.

```
for i in range(49):
  plt.subplot(7, 7, i+1)
  plt.axis("off")
  #plot raw pixel data
  plt.imshow(x_train[i])
plt.show()
```

إذا قمت بطباعة شكل مجموعة البيانات، فإن بيانات التدريب train data تحتوي على 60000 صورة بحجم 28*28 وبيانات الاختبار test data تحتوي على 10000 صورة بحجم 28*28.

تسطيح وتوسيع نطاق البيانات

نظرًا لأن أبعاد مجموعة البيانات هي 3، فسوف نقوم بتسطيحها إلى بعدين و28*28 تعني 684 وسيتم تحويلها إلى 60000في 684.

```
N, H, W = x_train.shape  #number, height, width
D = H * W   #dimension (28, 28)
x_train = x_train.reshape(-1, D)
x_test = x_test.reshape(-1, D)
```

تعريف نموذج المولد

نحدد هنا دالة لتطوير شبكة عصبية تلافيفية عميقة deep convolutional Neural network. البعد الكامن latent dimension هو متغير يحدد عدد المدخلات إلى النموذج. نعرف طبقة الإدخال input layer، ثلاث طبقات مخفية hidden layers تليها تسوية الدفعة Batch normalization، ودالة التنشيط باسم Leaky RELU وطبقة الإخراج مع دالة التنشيط مثل tanh لأن نطاق بكسل الصورة يتراوح بين ـ1 و+1.

```
# Defining Generator Model
latent_dim = 100
def build_generator(latent_dim):
  i = Input(shape=(latent_dim,))
  x = Dense(256, activation=LeakyReLU(alpha=0.2))(i)
  x = BatchNormalization(momentum=0.7)(x)
  x = Dense(512, activation=LeakyReLU(alpha=0.2))(x)
  x = BatchNormalization(momentum=0.7)(x)
  x = Dense(1024, activation=LeakyReLU(alpha=0.2))(x)
  x = BatchNormalization(momentum=0.7)(x)
  x = Dense(D, activation='tanh')(x)  #because Image pixel
is between -1 to 1.
  model = Model(i, x)  #i is input x is output layer
  return model
```

لماذا نستخدم Leaky RELU؟

يساعد Leaky relu على تدفق التدرج Gradient flow بسهولة عبر معمارية الشبكة العصبية.

- تأخذ دالة تنشيط ReLU فقط القيمة القصوى بين الإدخال والصفر. إذا استخدمنا ReLU، فمن المحتمل أن تتعطل الشبكةفي حالة تُعرف باسم Dying State. إذا حدث هذا فلن ينتج سوى الصفر لجميع المخرجات.
- هدفنا هو الحصول على قيمة التدرج من المميز لتشغيل المولد، وإذا تعطلت الشبكة، فلن يحدث التعلم.
- يستخدم Leaky ReLU معلمة تعرف باسم alpha للتحكمفي القيم السالبة ولا يتم تمرير الصفر مطلقًا. إذا كان الإدخال موجبًا، فسوف يظهر قيمة موجبة، وإذا استقبل سالبًا، فاضربه بألفا واسمح لبعض القيمة السالبة بالمرور عبر الشبكة.

$$f(x) = max(a \times x, x)$$

لماذا تسوية الدفعات؟

له تأثيرفي تثبيت عملية التدريب من خلال توحيد عمليات التنشيط من الطبقة السابقة بحيث يكون متوسطها صفرًا وتباينها واحداً. لقد أصبح تسوية الدُفعات Batch Normalization عنصرًا أساسيًا في حين أن تدريب الشبكات التلافيفية العميقة وشبكات GAN لا يختلف عنه.

أدى تطبيق معيار الدُفعة batch norm مباشرة على جميع الطبقات إلى تذبذب العينة وعدم استقرار النموذج.

تعريف نموذج المميز

نحن هنا نطور شبكة عصبية امامية التغذية Feed Forward Neural network بسيطة للمميز حيث سنمرر حجم الصورة. دالة التنشيط المستخدمة هي leaky ReLU وأنت تعرف السبب وراء ذلك ويتم استخدام sigmoidفي طبقة الإخراج لمشكلات التصنيف الثنائي لتصنيف الصور على أنها حقيقية أو مزيفة.

```
def build_discriminator(img_size):
  i = Input(shape=(img_size,))
  x = Dense(512, activation=LeakyReLU(alpha=0.2))(i)
  x = Dense(256, activation=LeakyReLU(alpha=0.2))(x)
  x = Dense(1, activation='sigmoid')(x)
  model = Model(i, x)
  return model
```

تجميع النماذج

حان الوقت الآن لتجميع compile المكونات المحددة لشبكات GAN.

```
# Build and compile the discriminator
discriminator = build_discriminator(D)
discriminator.compile ( loss='binary_crossentropy',
optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])
# Build and compile the combined model
generator = build_generator(latent_dim)
```

تمثيل عينة الضوضاء

سنقوم الآن بإنشاء مدخلات لتمثيل عينات الضوضاء من الفضاء الكامن latent space. ونمرر هذه الضوضاء إلى المولد لتوليد الصورة. بعد ذلك، نقوم بتمرير الصورة المولدة إلى المميز ونتوقع أنها مزيفة أم حقيقية.في المرحلة الأولية، لا نريد أن يتم تدريب المُميز وتكون الصورة مزيفة.

```
## Create an input to represent noise sample from latent
space
z = Input(shape=(latent_dim,))
## Pass noise through a generator to get an Image
img = generator(z)
discriminator.trainable = False
fake_pred = discriminator(img)
```

إنشاء نموذج المولد

لقد حان الوقت لإنشاء نموذج مولد مدمج مع مدخلات الضوضاء وملاحظات feedback المميز التي تساعد المولد على تحسين أدائه.

```
combined_model_gen = Model(z, fake_pred)  #first is noise
and 2nd is fake prediction
# Compile the combined model
combined_model_gen.compile(loss='binary_crossentropy',
optimizer=Adam(0.0002, 0.5))
```

تعريف معلمات تدريب GAN

حدد الفترات epochs وحجم الدفعة batch size وفترة العينة sample period مما يعني أنه بعد عدد الخطوات التي سيقوم المولد بإنشاء العينة. بعد ذلك، نحدد تسميات الدفعة Batch labels على أنها واحد وصفر. واحد يمثل أن الصورة حقيقية والصفر يمثل أن الصورة مزيفة. وقمنا أيضًا بإنشاء قائمتين فارغتين لتخزين اخطاء المولد والمميز. والأهم من ذلك أننا نقوم بإنشاء ملف فارغ في دليل العمل حيث سيتم حفظ الصورة التي تم إنشاؤها من خلال المولد.

```
batch_size = 32
epochs = 12000
sample_period = 200
ones = np.ones(batch_size)
zeros = np.zeros(batch_size)
#store generator and discriminator loss in each step or each
epoch
d_losses = []
```

```
g_losses = []
#create a file in which generator will create and save
images
if not os.path.exists('gan_images'):
  os.makedirs('gan_images')
```

دالة لإنشاء نماذج الصور

قم بإنشاء دالة تقوم بإنشاء شبكة من العينات العشوائية من المولد وحفظهافي ملف. بكلمات بسيطة، سيتم إنشاء صور عشوائيةفي بعض الفترات. نحدد حجم الصف بـ 5 والعمود أيضًا بـ 5، لذلكفي تكرار واحد أو على صفحة واحدة سيتم إنشاء 25 صورة.

```
def sample_images(epoch):
  rows, cols = 5, 5
  noise = np.random.randn(rows * cols, latent_dim)
  imgs = generator.predict(noise)
  # Rescale images 0 - 1
  imgs = 0.5 * imgs + 0.5
  fig, axs = plt.subplots(rows, cols)  #fig to plot img and
axis to store
  idx = 0
  for i in range(rows):  #5*5 loop means on page 25 imgs
will be there
    for j in range(cols):
      axs[i,j].imshow(imgs[idx].reshape(H, W), cmap='gray')
      axs[i,j].axis('off')
      idx += 1
  fig.savefig("gan_images/%d.png" % epoch)
  plt.close()
```

تدريب المميز ثم المولد لإنشاء الصور

الآن دعونا نبدأ بتدريب المميز. يتعين علينا تمرير صور حقيقية تعني مجموعة بيانات MNIST بالإضافة إلى بعض الصور المزيفة إلى المميز لتدريبها جيداً حتى تكون قادرة على تصنيف الصور. بعد ذلك، نقوم بإنشاء شبكة ضوضاء عشوائية مثل الصورة الحقيقية ونمررها إلى المولد لإنشاء صورة جديدة. بعد ذلك، نقوم بحساب خطأ كلا النموذجين وفي الصورة التي تم إنشاؤها، نقوم بتمرير التسمية كواحدة لخداع المميز للاعتقاد والتحقق من أنه قادر على التعرف عليها على أنها مزيفة أم لا.

```
#FIRST we will train Discriminator(with real imgs and fake
imgs)
# Main training loop
for epoch in range(epochs):
  ###########################
  ### Train discriminator ###
  ###########################
  # Select a random batch of images
  idx = np.random.randint(0, x_train.shape[0], batch_size)
  real_imgs = x_train[idx]  #MNIST dataset
```

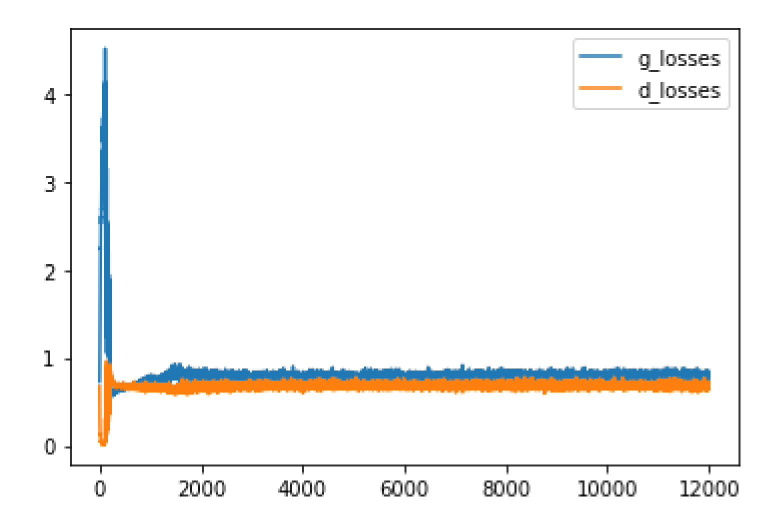
```
  # Generate fake images
  noise = np.random.randn(batch_size, latent_dim)
#generator to generate fake imgs
  fake_imgs = generator.predict(noise)
  # Train the discriminator
  # both loss and accuracy are returned
  d_loss_real, d_acc_real =
discriminator.train_on_batch(real_imgs, ones)  #belong to
positive class(real imgs)
  d_loss_fake, d_acc_fake =
discriminator.train_on_batch(fake_imgs, zeros)  #fake imgs
  d_loss = 0.5 * (d_loss_real + d_loss_fake)
  d_acc  = 0.5 * (d_acc_real + d_acc_fake)
  #######################
  ### Train generator ###
  #######################
  noise = np.random.randn(batch_size, latent_dim)
  g_loss = combined_model_gen.train_on_batch(noise, ones)
  #Now we are trying to fool the discriminator that generate
imgs are real that's why we are providing label as 1
  # do it again!
  noise = np.random.randn(batch_size, latent_dim)
  g_loss = combined_model_gen.train_on_batch(noise, ones)
  # Save the losses
  d_losses.append(d_loss)  #save the loss at each epoch
  g_losses.append(g_loss)
  if epoch % 100 == 0:
    print(f"epoch: {epoch+1}/{epochs}, d_loss: {d_loss:.2f},
      d_acc: {d_acc:.2f}, g_loss: {g_loss:.2f}")
  if epoch % sample_period == 0:
    sample_images(epoch)
```

قمنا بتدريبه على 12000 فترة، يمكنك التدرب على المزيد من الفترات. سيستغرق الأمر بعض الوقت، لذا من الأفضل استخدام Kaggle أو Google Colab GPU. وسيتم حفظ الصور التي تم إنشاؤها بالاسم gan image متبوعًا برقم الفترةفي الدليل المحدد.

رسم دالة الخطأ

لقد انتهينا من تدريب GAN ودعونا نرى مدى الدقة التي يستطيع المولد تحقيقهافي جعل المميز يُخدع.

```
plt.plot(g_losses, label='g_losses')
plt.plot(d_losses, label='d_losses')
plt.legend()
```



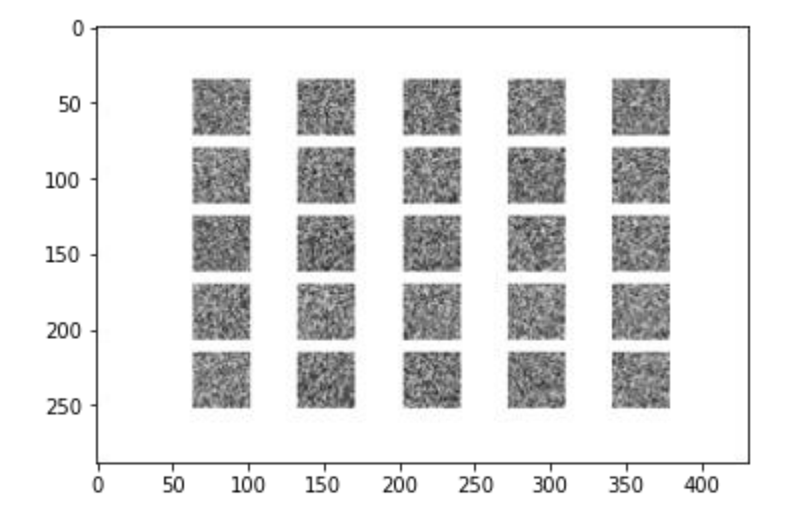
التحقق من النتائج

دعونا نرسم الصور التي تم إنشاؤهافي فترات مختلفة لنرى أنه بعد عدد الفترات التي تمكن المولد من استخراج بعض المعلومات.

رسم الصورة التي تم إنشاؤهافي فترة الصفر zero epoch

```
from skimage.io import imread
a = imread('gan_images/0.png')
plt.imshow(a)
```

دعونا نرىفي الفترة الأولى ما الذي يؤدي إلى إنشائه.

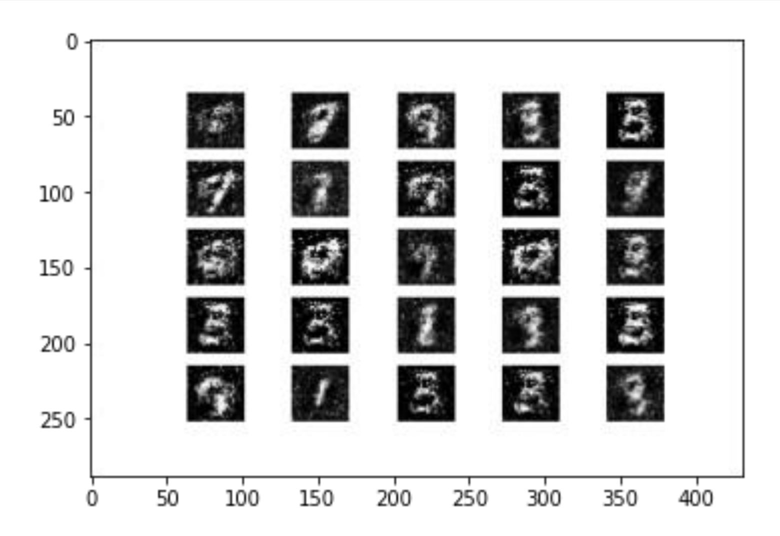


لا يتم استخراج أي معلومات من المولد ويكون المُميز ذكيًا بما يكفي للتعرف عليها على أنها مزيفة.



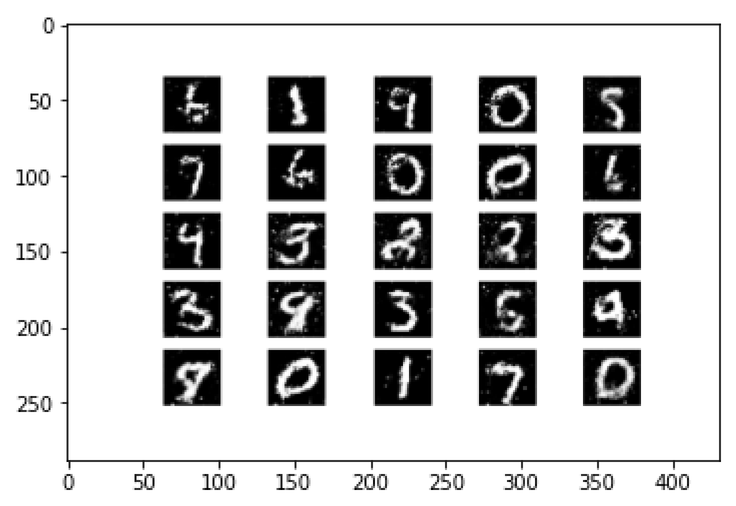
رسم الصورة التي تم إنشاؤها بعد التدريب على 1000 فترة

```
from skimage.io import imread
a = imread('gan_images/10000.png')
plt.imshow(a)
```



أصبح الآن المولد قادرًا ببطء على استخراج بعض المعلومات التي يمكن ملاحظتها.

رسم الصورة التي تم إنشاؤها بعد التدريب على 10000 فترة



الآن أصبح المولد قادرًا على البناء لأنه عبارة عن صورة لمجموعة بيانات MNIST وهناك فرص كبيرة لأن يكون المميز قد تم خداعه.

الاستنتاج

تهانينا، نحن قادرون على تنفيذ وفهم مفهوم وعمل شبكات GAN. تُعد شبكات GAN مجالًا جديدًا في مجال الذكاء الاصطناعي ويمكن أن يصدم الجميع بتطبيقاته ونتائجه. البحثفي هذا المجالفي ذروته وهناك المزيد من التطبيقات الجديدةفي هذا المجال على وشك أن تأتي. من الجيد جدًا أن تمتلك المعرفة العملية الأساسية لمثل هذه التقنيات الجديدةفي مجالنا. تتضمن مجالات التطبيقات والبحث القادمة لـ GAN إنشاء تصميمات داخلية مختلفة وصيغ جزيئية وألوان وما إلى ذلك. GAN هي تقنية توفر للآلات الخيال.

المصدر:

https://www.analyticsvidhya.com/blog/2021/10/an-end-to-end-introduction-to-generative-adversarial-networksgans/

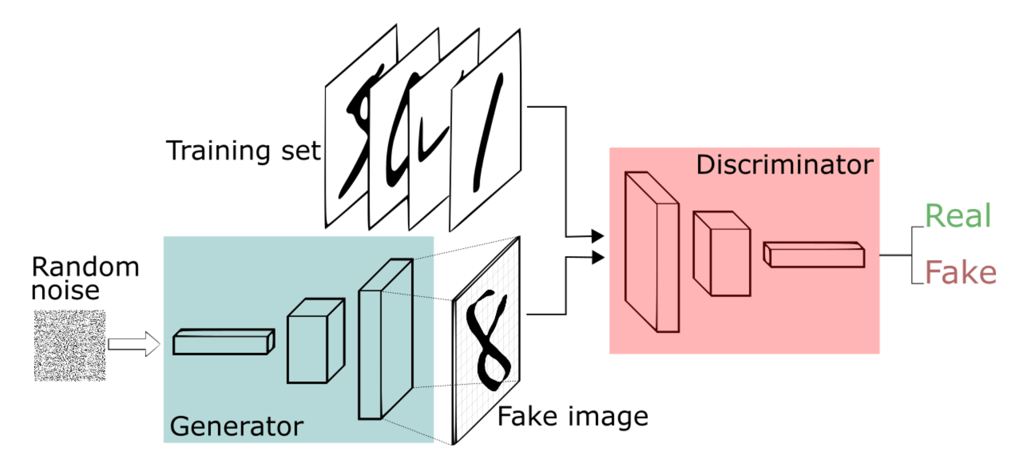
1) شبكات الخصومة التوليدية GANs Generative Adversarial Networks(GANs)

ما هي شبكات GAN؟

لقد سمعنا جميعًا عن شبكات RNN، التي تُستخدم أيضًا كمولد نص text generator ويمكنها إنشاء كلمة واحدة أو حرف واحدفي كل مرة. هل يمكن عمل هذا الشيء للصور؟ نعم، هناك تقنيات مثل شبكات الاعتقاد المرئية بالكامل Fully Visible Belief Networks (التي أعيدت تسميتها لاحقًا باسم نماذج التراجع التلقائي Auto-Regressive Models) التي تولد بكسلًا واحدًافي كل مرة والتي يمكن أن تؤدي إلى صورة تم إنشاؤها بالكامل (تبدو حقيقية ولكنها ليست كذلك). ولكن هل هناك أي طريقة لتوليد الصورة كاملةفي لقطة واحدة؟ الجواب هو نعم مرة أخرى. شبكات GAN هي النماذج المستخدمة لإنشاء صورة كاملةفي المرة الواحدة.

كيف تعمل شبكات GAN؟

تتكون شبكات GAN من مكونين مختلفين، المولد Generator والمميز Discriminator.في شبكات الخصومة التوليدية Generative Adversarial Networks، تعني كلمة Adversarial العكس أو بطريقة أخرى يتنافس المولد والمميز مع بعضهما البعض من أجل إنتاج صور واقعية.



المولد Generator عبارة عن شبكة عصبية تستخدم دالة قابلة للتفاضل differentiable function، فهي تأخذ ضوضاء عشوائية كمدخلات وتمرر تلك الضوضاء من خلال الدالة القابلة للتفاضل، وتحولها/تعيد تشكيلها لجعلها يمكن التعرف عليها. يحاول إنتاج صور حقيقية تعتمد كليًا على ضوضاء الإدخال input noises. لكن السؤال هو كيف ينتج هذا المولد الصورة الصحيحة؟ يجب تدريبه أولاً حتى يتمكن من إنتاج صور واقعية، أليس كذلك؟

تختلف عملية تدريب شبكات GAN عن تلك الخاصة بشبكات CNN، حيث نقوم بتدريب النموذج على صور متعددة جنبًا إلى جنب مع فئات الإخراج الخاصة بها،في شبكات GAN لا يوجد مخرجات مرتبطة بكل صورة إدخال. نعرض فقط على النموذج مجموعة من الصور ونطلب من النموذج إنتاج بعض الصور الجديدة التي تأتي من نفس التوزيع الاحتمالي.

لذلك نقوم بتمرير ضوضاء عشوائية random noises من خلال شبكة المولدات التي تنتج الصور الناتجة عن طريق تحديد الميزات من مجموعة الإدخال. هناك شبكة أخرى تستخدمها شبكات GAN تسمى المميز Discriminator، والتي توجه المولد إذا كانت الصورة المنتجة حقيقية أم لا. المميز عبارة عن شبكة عصبية عادية تقوم فقط بمهمة التصنيف. يتم تدريبه بنصف الصور الحقيقية ونصف الصور المزيفة (المولدة) حيث يتم تعيين احتمالية للصور الحقيقية بالقرب من 1 بينما يتم تعيين احتمالية للصور المزيفة بالقرب من 0.

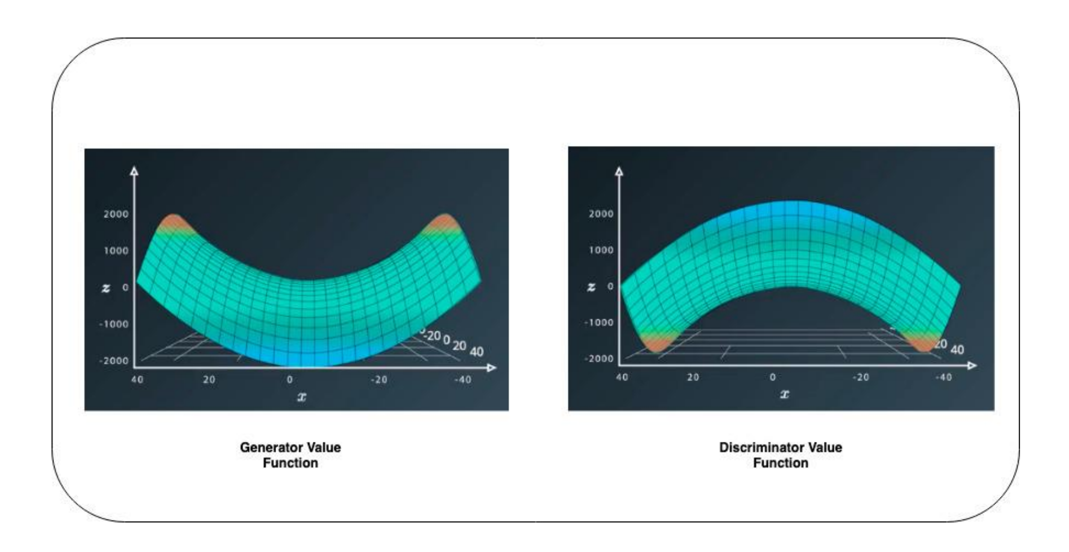
وفي الوقت نفسه، يتم تدريب المولد على العكس تمامًا، فهو يحاول إنشاء الصور حيث يقوم المميز بتعيينه إلى ما يقرب من 1. وعلى مدار فترة زمنية، يضطر المولد إلى إنتاج صور أكثر واقعية يمكن أن تخدع المميز بسهولة.

هل تدريب شبكات GAN مشابه لشبكات CNN؟

تحتوي شبكات CNN العادية على بعض دوال الخسارة/التكلفة loss/cost function التي تريد تقليلها من أجل تحسين الدقة، لذلك نقوم بتغيير المعلمات التي تؤدي إلى الأوزان المثالية. يشبه تدريب شبكات GAN إلى حد ما شبكة CNN ولكن مع بعض التغييرات الرئيسية، لدينا هنا دالتان للخسارة/التكلفة مرتبطتان بكل من المولد والمميز، ودوال الخسارة (الخطأ) هذه متعارضة مع بعضها البعض. يمكن تفسير عملية التدريب الكاملة لشبكات GAN من خلال دالة القيمة value function، حيث يريد المولد تقليل دالة القيمة الخاصة به، ويريد المميز تعظيم دالة القيمة الخاصة به.



يمكننا بسهولة فهم دالة القيمة باستخدام نقطة السرج Saddle Point.



عندما ترى دالة قيمة المولد Generator Value Function نصل إلى النقطة المثالية من خلال الذهاب إلى الحد الأدنى من السرج بينما بالنسبة إلى المميز نحقق نفس الشيء من خلال الذهاب إلى أعلى (الذروة القصوى maximum peak) من السرج العكسي. وهذا ما يفسر أنه بالنسبة لتدريب شبكات GAN، نحتاج إلى تقنيتين للتحسين لكل من المولد والمميز، اللتين ستعملان في وقت واحد.

لا توجد طبقة التفافية Convolution أو متكررة Recurrent مستخدمة للتدريب، إنها مجرد مضاعفة للمصفوفة متبوعة بدوال التنشيط (يفضل استخدام leaky relu وtanh وsigmoid). للحصول على تدريب أفضل، يجب أن يكون لدينا طبقة مخفية واحدة على الأقل لكل من المولد والمميز. والمُحسِّن المفضل لشبكات GAN هو مُحسِّن Adam.

تطبيقات شبكات GAN

يتم استخدام شبكات GAN في عدة أماكن، حاليًا، يتم استخدامها فقط كنشاط ممتع ولكن يمكن رؤية استخدام أكثر جدية في المستقبل. حاليًا، تُستخدم شبكات GAN في حالات الاستخدام التالية:

1. **إنشاء شخصيات كرتونية Generate Cartoon Characters:** باستخدام شبكات GAN، يمكن للمرء إنشاء شخصيات كرتونية بمجرد تمرير أصوات عشوائية، وتبدو تلك الشخصيات مثل تلك التي يصنعها مصمم الرسومات — https://github.com/mnicnc404/CartoonGan-tensorflow
2. **ترجمة الصورة إلى صورة Image to Image Translation:** استخدام صورة عادية واحدة لإنشاء اللوحات الكرتونية/ الفنية أو العكس — https://github.com/phillipi/pix2pix

3. **شيخوخة الوجه Face Aging**: هذا شيء ربما رأيته كثيرًا على تطبيقات الوسائط الاجتماعية، حيث يمكن للمرء تحويل الصور إلى صور أصغر أو أكبر سنًا — https://github.com/ZZUTK/Face-Aging-CAAE
4. **تلوين الصور Image Colourisation**: باستخدام شبكات GAN، يتم تحويل الصور بالأبيض والأسود إلى صور ملونة — https://github.com/jantic/DeOldify
5. **الصورة إلى فيديو مباشر Image to Live Video**: يمكنه تحويل الصور الثابتة إلى مقاطع فيديو قصيرة باستخدام بعض التعديلات الخوارزمية.
6. **إنشاء كائنات ثلاثية الأبعاد 3D object Generation**: تمامًا مثل الصور ثنائية الأبعاد، تستطيع شبكات GAN أيضًا إنشاء مقاطع فيديو ثلاثية الأبعاد.

معظم عمالقة التكنولوجيا (مثل Google، وMicrosoft، وAmazon، وما إلى ذلك) يعملون بجد على تطبيق شبكات GAN للاستخدام العملي، وبعض حالات الاستخدام هذه هي:

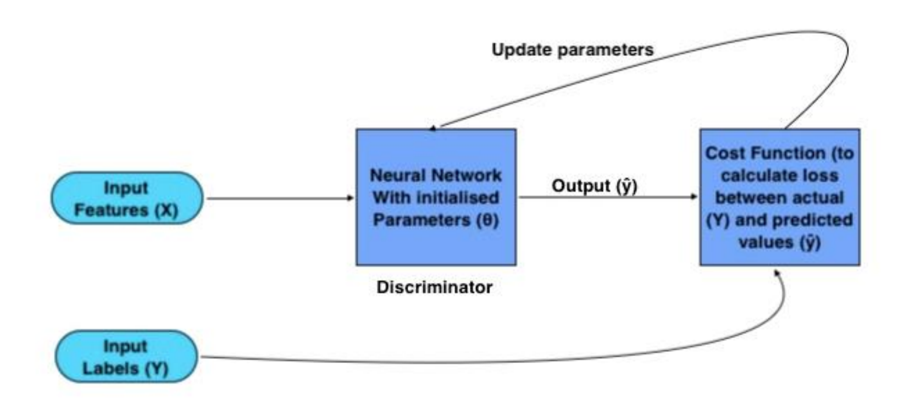
- **Adobe**: استخدام شبكات GAN للجيل التالي من Photoshop.
- **Google**: استخدام شبكات GAN لإنشاء النص.
- IBM: استخدام شبكات GAN لزيادة البيانات Data Augmentation (لإنشاء صور تركيبية synthetic images لتدريب نماذج التصنيف الخاصة بها).
- **Snap Chat/ TikTok**: لإنشاء فلاتر صور متنوعة (التي ربما تكون قد شاهدتها بالفعل).
- **Disney**: استخدام شبكات GAN للدقة الفائقة (تحسين جودة الفيديو) لأفلامهم.

الشيء المميزفي شبكات GAN هو أن هذه الشركات تعتمد عليهافي مستقبلها، ألا تعتقد ذلك؟

إذن ما الذي يمنعك من التعرف على هذه التكنولوجيا الملحمية؟ سأجيب عليه، لا شيء، أنت فقط بحاجة إلى البداية وهذه المقالة ستفعل ذلك. دعونا أولاً نناقش الرياضيات وراء المولد والمميز.

الوظيفة الرياضية للمميز

الغرض الوحيد من المميز هو تصنيف الصور الحقيقية والمزيفة. للتصنيف، يتم استخدام الشبكة العصبية التلافيفية التقليدية Convolutional Neural Network (CNN) مع دالة تكلفة cost function محددة. تعمل عملية التدريب الخاصة بالمميز على النحو التالي:



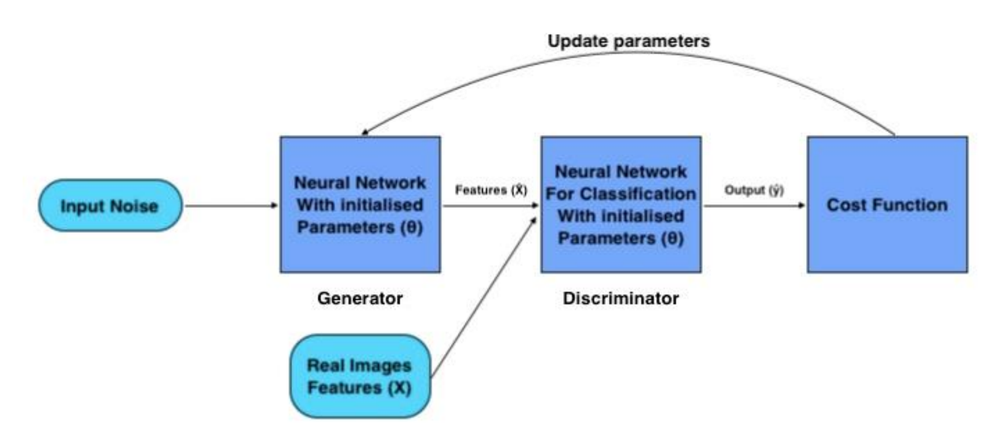
حيث أن X وY هما ميزات الإدخال input features والتسميات labels على التوالي، يتم تمثيل الإخراج باستخدام (ŷ) ويتم تمثيل معلمات الشبكة بواسطة (θ).

تحتاج شبكات GAN التدريبية إلى مجموعة من صور التدريب والتسميات الخاصة بها، وتنتقل هذه الصور كميزة إدخال إلى CNN، مع وجود مجموعة من المعلمات التي تمت تهيئتها. تقوم شبكة CNN هذه بإنشاء مخرجات عن طريق ضرب مصفوفة الوزن (W) مع ميزات الإدخال (X) وإضافة انحياز (B) فيها وتحويلها إلى مصفوفة غير خطية عن طريق تمريرها إلى دالة التنشيط activation function.

ويشار إلى هذا الإخراج على أنه مخرجات متوقعة predicted output، ثم يتم حساب الخطأ بناءً على معلمات الأوزان التي يتم ضبطهافي الشبكة من أجل تقليل الخطأ.

الوظيفة الرياضية للمولد

هدف المولد هو إنشاء صورة مزيفة من التوزيع المحدد (مجموعة الصور)، وهو يفعل ذلك من خلال الإجراء التالي:



يتم تمرير مجموعة من متجهات الإدخال input vectors (ضوضاء عشوائية random noise) عبر الشبكة العصبية للمولد والتي تقوم بإنشاء صورة جديدة تمامًا عن طريق ضرب مصفوفة وزن المولد مع ضوضاء الإدخال.

تعمل هذه الصورة التي تم إنشاؤها كمدخل للمميز الذي تم تدريبه لتصنيف الصور المزيفة والحقيقية. ثم يتم حساب الخطأ للصور التي تم إنشاؤها، وعلى أساسها يتم تحديث المعلمات للمولد حتى نحصل على دقة جيدة.

بمجرد أن نكون راضين عن دقة المولد، نقوم بحفظ أوزان المولد وإزالة المميز من الشبكة، واستخدام مصفوفة الوزن تلك لتوليد المزيد من الصور الجديدة عن طريق تمرير مصفوفة ضوضاء عشوائية مختلفةفي كل مرة.

خطأ الإنتروبيا المتقاطعة الثنائية لشبكات GAN

من أجل تحسين معلمات شبكات GAN، نحتاج إلى دالة خطأ تخبر الشبكة بمدى حاجتها للتحسين من خلال حساب الفرق بين القيمة الفعلية actual والمتوقعة predicted. تسمى دالة الخطأ المستخدمةفي شبكات GAN باسم الانتروبيا المتقاطعة الثنائية Binary Cross-Entropy ويتم تمثيلها على النحو التالي:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log h(x^{(i)},\theta) + (1-y^{(i)})\log(1-h(x^{(i)},\theta))]$$

حيث m هو حجم الدفعة، وy(i) هي قيمة التسمية الفعلية، وh هي قيمة التسمية المتوقعة، وx(i) هي ميزة الإدخال، وتمثل θ المعلمة.

دعونا نقسم دالة التكلفة هذه إلى أجزاء فرعية من أجل الحصول على فهم أفضل. الصيغة المعطاة هي مزيج من حدين حيث يتم استخدام أحدهما عندما يكون فعالاً عندما تكون التسمية "0" والآخر مهم عندما تكون التسمية "1". الحد الأول هو:

$$y^{(i)}\log h(x^{(i)},\theta)$$

إذا كانت القيمة الفعلية هي "1" والقيمة المتوقعة هي "0~"في هذه الحالة، نظرًا لأن log(~0) يميل إلى اللانهاية السالبة أو مرتفع جداً، وإذا كانت القيمة المتوقعة أيضًا "1~" فإن log(~1) سيكون قريبًا من "0" أو أقل جدًا، لذلك يساعد هذا الحدفي حساب الخطأ لقيم التسمية "1".

$$(1-y^{(i)})\log(1-h(x^{(i)},\theta))$$

إذا كانت القيمة الفعلية هي "0" والقيمة المتوقعة هي "1~"، فإن ((1~)-log(1 سيؤدي إلى لانهاية سالبة أو عالية جداً، وإذا كانت القيمة المتوقعة هي "0~" فإن المصطلح سيكون تنتج نتائج "0~" أو خطأ أقل جداً، لذلك يتم استخدام هذا الحد لقيم التسمية الفعلية "0".

سيعيد أي من حدي الخطأ القيم السالبةفي حالة خطأ التنبؤ، ويشار إلى مجموعة هذه الحدود باسم Entropy (Log Loss). ولكن بما أنها سلبية، لجعلها أكبر من "1" فإننا نطبق عليها إشارة سالبة (يمكنك أن ترىفي الصيغة الرئيسية)، وتطبيق هذه الإشارة السلبية هو ما يجعلها إنتروبيا متقاطعة Cross-Entropy (Negative Log Loss).

تدريب نموذج GAN الأول

سنقوم بإنشاء نموذج GAN يكون قادرًا على إنشاء أرقام مكتوبة بخط اليد من توزيع بيانات MNIST باستخدام وحدة PyTorch.

أولاً، لنستورد الوحدات المطلوبة:

```
%matplotlib inline
import numpy as np
import torch
import matplotlib.pyplot as plt
```

ثم نقرأ البيانات من الوحدة الفرعية التي توفرها PyTorch نفسها والتي تسمى datasets.

```
# number of subprocesses to use for data loading
num_workers = 0
# how many samples per batch to load
batch_size = 64
# convert data to torch.FloatTensor
transform = transforms.ToTensor()
# get the training datasets
train_data = datasets.MNIST(root='data', train=True,
                                   download=True,
transform=transform)
# prepare data loader
train_loader = torch.utils.data.DataLoader(train_data,
batch_size=batch_size,

num_workers=num_workers)
```

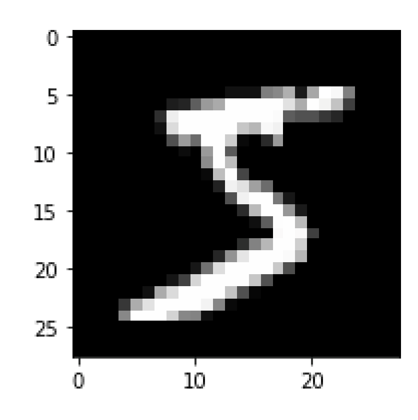
تصور البيانات

نظرًا لأننا سنقوم بإنشاء نموذجنا على إطار عمل PyTorch الذي يستخدم الموترات، فسنقوم بتحويل بياناتنا إلى torch tensors. إذا كنت تريد تصور البيانات، يمكنك المضي قدمًا واستخدام مجموعة التعليمات البرمجية التالية:

```
# obtain one batch of training images
```

```
dataiter = iter(train_loader)
images, labels = dataiter.next()
images = images.numpy()
# get one image from the batch
img = np.squeeze(images[0])
fig = plt.figure(figsize = (3,3))
ax = fig.add_subplot(111)
ax.imshow(img, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x10bab29e8>
```



المميز

حان الوقت الآن لتعريف شبكة المميز وهي عبارة عن مزيج من طبقات CNN المختلفة.

```
import torch.nn as nn
import torch.nn.functional as F
class Discriminator(nn.Module):
    def __init__(self, input_size, hidden_dim, output_size):
        super(Discriminator, self).__init__()
        # define hidden linear layers
        self.fc1 = nn.Linear(input_size, hidden_dim*4)
        self.fc2 = nn.Linear(hidden_dim*4, hidden_dim*2)
        self.fc3 = nn.Linear(hidden_dim*2, hidden_dim)
        # final fully-connected layer
        self.fc4 = nn.Linear(hidden_dim, output_size)
        # dropout layer
        self.dropout = nn.Dropout(0.3)
    def forward(self, x):
        # flatten image
        x = x.view(-1, 28*28)
        # all hidden layers
        x = F.leaky_relu(self.fc1(x), 0.2) # (input,
negative_slope=0.2)
        x = self.dropout(x)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = self.dropout(x)
```

```
        x = F.leaky_relu(self.fc3(x), 0.2)
        x = self.dropout(x)
        # final layer
        out = self.fc4(x)
        return out
```

يتبع الكود أعلاه معمارية Python التقليدية القائمة على الكائنات. fc1، fc2، fc3، fc3 هي الطبقات المتصلة بالكامل fully connected layers. عندما نمرر ميزات الإدخال input features الخاصة بنا، فإنها تمر عبر كل هذه الطبقات بدءًا من fc1، وفي النهاية، لدينا طبقة واحدة dropout layer تُستخدم لمعالجة مشكلة الضبط الزائد overfitting.

في نفس الكود، ستشاهد دالة باسم Forward (self, x)، هذه الدالة هي تنفيذ آلية الانتشار الأمامي forward propagation الفعلية حيث تتبع كل طبقة (fc1، fc2، fc3، وfc4) دالة التنشيط (leaky_relu) لتحويل المخرجات الخطية إلى غير خطية

المولد

بعد ذلك سوف نتحقق من مقطع المولدفي GAN:

```
class Generator(nn.Module):
    def __init__(self, input_size, hidden_dim, output_size):
        super(Generator, self).__init__()
        # define hidden linear layers
        self.fc1 = nn.Linear(input_size, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim*2)
        self.fc3 = nn.Linear(hidden_dim*2, hidden_dim*4)
        # final fully-connected layer
        self.fc4 = nn.Linear(hidden_dim*4, output_size)
        # dropout layer
        self.dropout = nn.Dropout(0.3)
    def forward(self, x):
        # all hidden layers
        x = F.leaky_relu(self.fc1(x), 0.2) # (input,
negative_slope=0.2)
        x = self.dropout(x)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = self.dropout(x)
        x = F.leaky_relu(self.fc3(x), 0.2)
        x = self.dropout(x)
        # final layer with tanh applied
        out = F.tanh(self.fc4(x))
        return out
```

تم إنشاء شبكة المولدات أيضًا من الطبقات المتصلة بالكامل، ودوال تنشيط leaky relu، والتسرب dropout. الشيء الوحيد الذي يجعله مختلفًا عن المميز هو أنه يعطي مخرجات اعتمادًا على معلمة input_size (وهو حجم الصورة المراد إنشاؤها).

ضبط المعلمات الفائقة

المعلمات الفائقة Hyperparameters التي سنستخدمها لتدريب شبكات GAN هي:

```
# Discriminator hyperparams
# Size of input image to discriminator (28*28)
input_size = 784
# Size of discriminator output (real or fake)
d_output_size = 1
# Size of last hidden layer in the discriminator
d_hidden_size = 32
# Generator hyperparams
# Size of latent vector to give to generator
z_size = 100
# Size of discriminator output (generated image)
g_output_size = 784
# Size of first hidden layer in the generator
g_hidden_size = 32
```

إنشاء مثيل للنماذج

وأخيرًا، ستبدو الشبكة الكاملة كما يلي:

```
# instantiate discriminator and generator
D = Discriminator(input_size, d_hidden_size, d_output_size)
G = Generator(z_size, g_hidden_size, g_output_size)
# check that they are as you expect
print(D)
print( )
print(G)
```

```
Discriminator(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=32, bias=True)
  (fc4): Linear(in_features=32, out_features=1, bias=True)
  (dropout): Dropout(p=0.3)
)

Generator(
  (fc1): Linear(in_features=100, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=128, bias=True)
  (fc4): Linear(in_features=128, out_features=784, bias=True)
  (dropout): Dropout(p=0.3)
)
```

حساب الاخطاء

لقد قمنا بتعريف المولد والمميز والآن حان الوقت لتعريف اخطاءهما حتى تتحسن تلك الشبكات بمرور الوقت. بالنسبة لشبكات GAN، سيكون لدينا دالة خطأ حقيقي real loss وخطأ مزيف fake loss والتي سيتم تعريفها على النحو التالي:

```
# Calculate losses
def real_loss(D_out, smooth=False):
    batch_size = D_out.size(0)
    # label smoothing
    if smooth:
        # smooth, real labels = 0.9
        labels = torch.ones(batch_size)*0.9
    else:
        labels = torch.ones(batch_size) # real labels = 1
    # numerically stable loss
    criterion = nn.BCEWithLogitsLoss()
    # calculate loss
    loss = criterion(D_out.squeeze(), labels)
    return loss
def fake_loss(D_out):
    batch_size = D_out.size(0)
    labels = torch.zeros(batch_size) # fake labels = 0
    criterion = nn.BCEWithLogitsLoss()
    # calculate loss
    loss = criterion(D_out.squeeze(), labels)
    return loss
```

المحسنات

بمجرد تعريف الاخطاء، سنختار مُحسِّنًا optimizer مناسبًا للتدريب:

```
import torch.optim as optim
# Optimizers
lr = 0.002
# Create optimizers for the discriminator and generator
d_optimizer = optim.Adam(D.parameters(), lr)
g_optimizer = optim.Adam(G.parameters(), lr)
```

تدريب النماذج

نظرًا لأننا قمنا بتعريف المولد والمميز لكل من الشبكات ودوال الخطأ والمحسنات، فإننا الآن نستخدم الفترات epochs والميزات features الأخرى لتدريب الشبكة بأكملها.

```
import pickle as pkl
# training hyperparams
num_epochs = 100
# keep track of loss and generated, "fake" samples
samples = []
losses = []
```

```
print_every = 400
# Get some fixed data for sampling. These are images that
are held
# constant throughout training, and allow us to inspect the
model's performance
sample_size=16
fixed_z = np.random.uniform(-1, 1, size=(sample_size,
z_size))
fixed_z = torch.from_numpy(fixed_z).float()
# train the network
D.train()
G.train()
for epoch in range(num_epochs):
    for batch_i, (real_images, _) in
enumerate(train_loader):
        batch_size = real_images.size(0)
        ## Important rescaling step ##
        real_images = real_images*2 - 1  # rescale input
images from [0,1) to [-1, 1)
        # ============================================
        #            TRAIN THE DISCRIMINATOR
        # ============================================
        d_optimizer.zero_grad()
        # 1. Train with real images
        # Compute the discriminator losses on real images
        # smooth the real labels
        D_real = D(real_images)
        d_real_loss = real_loss(D_real, smooth=True)
        # 2. Train with fake images
        # Generate fake images
        # gradients don't have to flow during this step
        with torch.no_grad():
            z = np.random.uniform(-1, 1, size=(batch_size,
z_size))
            z = torch.from_numpy(z).float()
            fake_images = G(z)
        # Compute the discriminator losses on fake images
        D_fake = D(fake_images)
        d_fake_loss = fake_loss(D_fake)
        # add up loss and perform backprop
        d_loss = d_real_loss + d_fake_loss
        d_loss.backward()
        d_optimizer.step()
        # =========================================
        #            TRAIN THE GENERATOR
        # =========================================
        g_optimizer.zero_grad()
        # 1. Train with fake images and flipped labels
        # Generate fake images
```

```
        z = np.random.uniform(-1, 1, size=(batch_size,
z_size))
        z = torch.from_numpy(z).float()
        fake_images = G(z)
        # Compute the discriminator losses on fake images
        # using flipped labels!
        D_fake = D(fake_images)
        g_loss = real_loss(D_fake) # use real loss to flip
labels
        # perform backprop
        g_loss.backward()
        g_optimizer.step()
        # Print some loss stats
        if batch_i % print_every == 0:
            # print discriminator and generator loss
            print('Epoch [{:5d}/{:5d}] | d_loss: {:6.4f} |
g_loss: {:6.4f}'.format(
                    epoch+1, num_epochs, d_loss.item(),
g_loss.item()))
    ## AFTER EACH EPOCH##
    # append discriminator loss and generator loss
    losses.append((d_loss.item(), g_loss.item()))
    # generate and save sample, fake images
    G.eval() # eval mode for generating samples
    samples_z = G(fixed_z)
    samples.append(samples_z)
    G.train() # back to train mode
# Save training generator samples
with open('train_samples.pkl', 'wb') as f:
    pkl.dump(samples, f)
```

بمجرد تشغيل مجموعة التعليمات البرمجية أعلاه، ستبدأ عملية التدريب على النحو التالي:

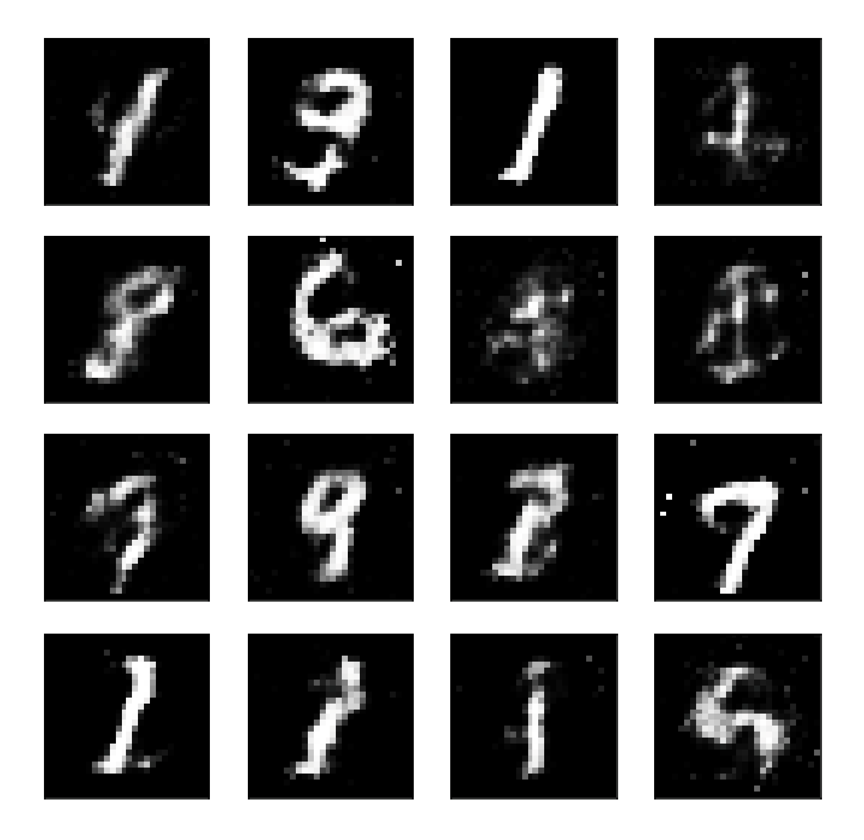
```
Epoch [    1/  100] | d_loss: 1.3830 | g_loss: 0.6883
Epoch [    1/  100] | d_loss: 0.7668 | g_loss: 3.1404
Epoch [    1/  100] | d_loss: 1.0908 | g_loss: 2.6393
Epoch [    2/  100] | d_loss: 1.2982 | g_loss: 1.1008
Epoch [    2/  100] | d_loss: 1.1753 | g_loss: 1.1042
Epoch [    2/  100] | d_loss: 1.0874 | g_loss: 1.6588
Epoch [    3/  100] | d_loss: 0.9747 | g_loss: 2.9083
Epoch [    3/  100] | d_loss: 1.0724 | g_loss: 1.1101
Epoch [    3/  100] | d_loss: 1.4727 | g_loss: 0.4990
Epoch [    4/  100] | d_loss: 1.1590 | g_loss: 0.9996
Epoch [    4/  100] | d_loss: 1.1832 | g_loss: 1.0638
Epoch [    4/  100] | d_loss: 1.1652 | g_loss: 1.1463
Epoch [    5/  100] | d_loss: 1.2966 | g_loss: 1.0950
Epoch [    5/  100] | d_loss: 1.3616 | g_loss: 1.0312
Epoch [    5/  100] | d_loss: 1.2202 | g_loss: 1.1301
Epoch [    6/  100] | d_loss: 1.1609 | g_loss: 1.5148
Epoch [    6/  100] | d_loss: 1.1100 | g_loss: 1.2183
Epoch [    6/  100] | d_loss: 1.2376 | g_loss: 1.5296
Epoch [    7/  100] | d_loss: 1.2304 | g_loss: 1.1526
```

توليد الصور

وأخيرًا، عندما يتم تدريب النموذج، يمكنك استخدام المولد المُدرب لإنتاج الصور المكتوبة بخط اليد الجديدة.

```
# randomly generated, new latent vectors
sample_size=16
rand_z = np.random.uniform(-1, 1, size=(sample_size,
z_size))
rand_z = torch.from_numpy(rand_z).float()
G.eval() # eval mode
# generated samples
rand_images = G(rand_z)
# 0 indicates the first set of samples in the passed in list
# and we only have one batch of samples, here
view_samples(0, [rand_images])
```

إن الإخراج الذي تم إنشاؤه باستخدام الكود التالي يريد شيئًا مثل هذا:



والآن لديك نموذج GAN المدرب الخاص بك، ويمكنك استخدام هذا النموذج لتدريبه على مجموعة مختلفة من الصور، لإنتاج صور جديدة غير مرئية.

المصدر:

https://www.analyticsvidhya.com/blog/2021/04/lets-talk-about-gans/

https://www.analyticsvidhya.com/blog/2021/05/%e2%80%8atrain-your-first-gan-model-lets-talk-about-gans-part-2%e2%80%8a/

2) شبكات الخصومة التوليدية: بناء نماذجك الأولى Generative Adversarial Networks: Build Your First Models

شبكات الخصومة التوليدية Generative adversarial networks (GANs) هي شبكات عصبية تولد مواد، مثل الصور أو الموسيقى أو الكلام أو النصوص، تشبه ما ينتجه البشر.

لقد كانت شبكات GAN موضوعًا نشطًا للبحث في السنوات الأخيرة. ووصف يان ليكون، مدير أبحاث الذكاء الاصطناعي في فيسبوك، التدريب التنافسي adversarial training بأنه "الفكرة الأكثر إثارة للاهتمام في السنوات العشر الماضية" في مجال التعلم الآلي. أدناه، ستتعرف على كيفية عمل شبكات GAN قبل تنفيذ نموذجين توليديين خاصين بك.

في هذا البرنامج التعليمي، ستتعلم:

- ما هو النموذج التوليدي generative model وكيف يختلف عن النموذج التمييزي discriminative model؟
- كيف يتم تنظيم شبكات GAN وتدريبها؟
- كيفية إنشاء شبكة GAN الخاصة بك باستخدام PyTorch؟
- كيفية تدريب GAN الخاص بك على التطبيقات العملية باستخدام GPU وPyTorch

هيا بنا نبدأ!

ما هي شبكات الخصومة التوليدية؟

شبكات الخصومة التوليدية هي أنظمة للتعلم الآلي يمكنها تعلم محاكاة توزيع معين للبيانات. تم اقتراحها لأول مرة في ورقة بحثية نشرتها شركة NeurIPS عام 2014 من قبل خبير التعلم العميق إيان جودفيلو وزملائه.

تتكون شبكات GAN من شبكتين عصبيتين، إحداهما مدربة على توليد البيانات والأخرى مدربة على التمييز بين البيانات المزيفة fake data والبيانات الحقيقية real data (ومن هنا تأتي الطبيعة "العدائية adversarial" للنموذج). على الرغم من أن فكرة إنشاء معمارية لتوليد البيانات ليست جديدة، عندما يتعلق الأمر بتوليد الصور والفيديو، فقد قدمت شبكات GAN نتائج مبهرة مثل:

- **نقل النمط Style transfer باستخدام** CycleGAN، والذي يمكنه إجراء عدد من تحويلات النمط المقنعة على الصور.

- توليد الوجوه البشرية باستخدام StyleGAN، كما هو موضح في موقع الويب " This Person Does Not Exist".

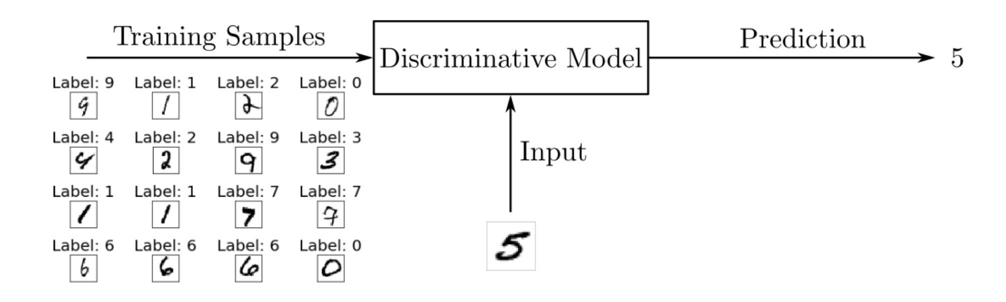
تعتبر الهياكل التي تولد البيانات، بما في ذلك شبكات GAN، النماذج التوليدية generative models على عكس النماذج التمييزية discriminative models التي تمت دراستها على نطاق أوسع. قبل الغوص في شبكات GAN، ستنظر إلى الاختلافات بين هذين النوعين من النماذج.

النماذج التمييزية مقابل النماذج التوليدية

إذا كنت قد درست الشبكات العصبية، فمن المحتمل أن معظم التطبيقات التي صادفتك تم تنفيذها باستخدام نماذج تمييزية. من ناحية أخرى، تعد شبكات الخصومة التوليدية جزءًا من فئة مختلفة من النماذج المعروفة باسم النماذج التوليدية.

النماذج التمييزية هي تلك المستخدمة في معظم مشكلات التصنيف classification أو الانحدار regression الخاضعة للإشراف supervised. كمثال على مشكلة التصنيف، لنفترض أنك ترغب في تدريب نموذج لتصنيف صور الأرقام المكتوبة بخط اليد من 0 إلى 9. ولهذا السبب، يمكنك استخدام مجموعة بيانات ذات علامات تحتوي على صور للأرقام المكتوبة بخط اليد والتسميات المرتبطة بها تشير إلى كل رقم تمثل الصورة.

أثناء عملية التدريب، يمكنك استخدام خوارزمية لضبط معلمات النموذج. سيكون الهدف هو تقليل دالة الخسارة (الخطأ) loss function بحيث يتعلم النموذج التوزيع الاحتمالي للمخرجات بالنظر إلى المدخلات. بعد مرحلة التدريب، يمكنك استخدام النموذج لتصنيف صورة أرقام جديدة مكتوبة بخط اليد من خلال تقدير الرقم الأكثر احتمالاً الذي يتوافق معه الإدخال، كما هو موضح في الشكل أدناه:

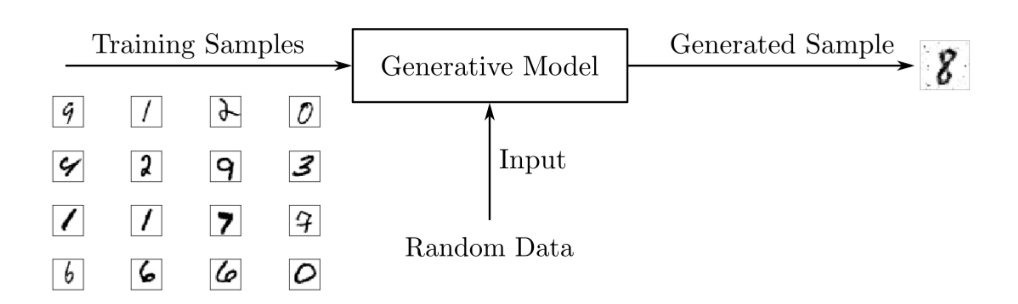


يمكنك تصوير النماذج التمييزية لمشاكل التصنيف على أنها كتل تستخدم بيانات التدريب لمعرفة الحدود بين الفئات. ثم يستخدمون هذه الحدود لتمييز المدخلات والتنبؤ بفئتها. من الناحية الرياضية، تتعلم النماذج التمييزية الاحتمال الشرطي $P(y|x)$ للمخرج y بالنظر إلى المدخلات x.

إلى جانب الشبكات العصبية، يمكن استخدام هياكل أخرى كنماذج تمييزية مثل نماذج الانحدار اللوجستي logistic regression models وآلات المتجهات الداعمة support vector machines (SVMs).

ومع ذلك، يتم تدريب النماذج التوليدية مثل شبكات GAN لوصف كيفية إنشاء مجموعة البيانات من حيث النموذج الاحتمالي. من خلال أخذ العينات من نموذج توليدي، يمكنك إنشاء بيانات جديدة.في حين يتم استخدام النماذج التمييزية للتعلم الخاضع للإشراف، غالبًا ما تستخدم النماذج التوليدية مع مجموعات البيانات غير المسماة unlabeled datasets ويمكن اعتبارها شكلاً من أشكال التعلم غير الخاضع للإشراف unsupervised learning.

باستخدام مجموعة بيانات الأرقام المكتوبة بخط اليد handwritten digits، يمكنك تدريب نموذج توليدي لإنشاء أرقام جديدة. أثناء مرحلة التدريب، يمكنك استخدام بعض الخوارزميات لضبط معلمات النموذج لتقليل دالة الخطأ ومعرفة التوزيع الاحتمالي لمجموعة التدريب. وبعد ذلك، باستخدام النموذج الذي تم تدريبه، يمكنك إنشاء عينات جديدة، كما هو موضح في الشكل التالي:



لإخراج عينات جديدة، عادة ما تأخذ النماذج التوليديةفي الاعتبار عنصرًا تصادفياً stochastic أو عشوائيًا random يؤثر على العينات التي يولدها النموذج. يتم الحصول على العينات العشوائية المستخدمة لقيادة المولد من مساحة كامنة تمثل فيها المتجهات نوعًا من الشكل المضغوط للعينات المولدة.

على عكس النماذج التمييزية، تتعلم النماذج التوليدية احتمالية P(x) لبيانات الإدخال x، ومن خلال توزيع بيانات الإدخال، تكون قادرة على إنشاء مثيلات بيانات جديدة.

ملاحظة: يمكن أيضًا استخدام النماذج التوليدية مع مجموعات البيانات المسماة labeled datasets. عندما يتم ذلك، يتم تدريبهم على معرفة احتمالية P(x|y) للمدخل x بالنظر إلى المخرج y. ويمكن

استخدامها أيضًا لمهام التصنيف، ولكن بشكل عام، تعمل النماذج التمييزية بشكل أفضل عندما يتعلق الأمر بالتصنيف.

على الرغم من أن شبكات GAN قد حظيت بالكثير من الاهتمام في السنوات الأخيرة، إلا أنها ليست المعمارية الوحيدة التي يمكن استخدامها كنموذج توليدي. إلى جانب شبكات GAN، هناك العديد من معماريات النماذج التوليدية الأخرى مثل:

- آلات بولتزمان Boltzmann machines.
- شبكات الترميز التلقائي المتغيرة Variational autoencoders.
- نماذج ماركوف المخفية Hidden Markov models.
- النماذج التي تتنبأ بالكلمة التالية في التسلسل، مثل GPT-2

ومع ذلك، فقد اجتذبت شبكات GAN الاهتمام العام الأكبر في الآونة الأخيرة بسبب النتائج المثيرة في توليد الصور والفيديو.

الآن بعد أن تعرفت على أساسيات النماذج التوليدية، سترى كيف تعمل شبكات GAN وكيفية تدريبها.

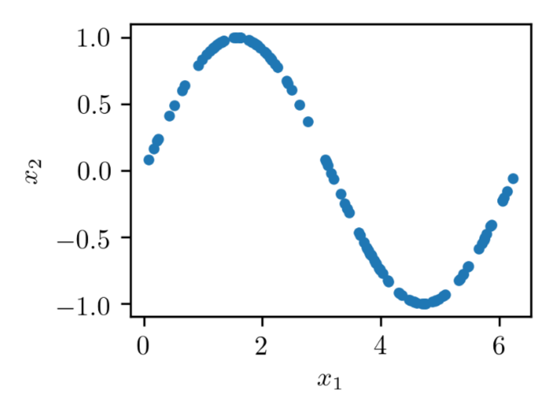
معمارية شبكات الخصومة التوليدية

تتكون الشبكات الخصومة التوليدية من معمارية شاملة مكونة من شبكتين عصبيتين، إحداهما تسمى المولد generator والأخرى تسمى المميز discriminator.

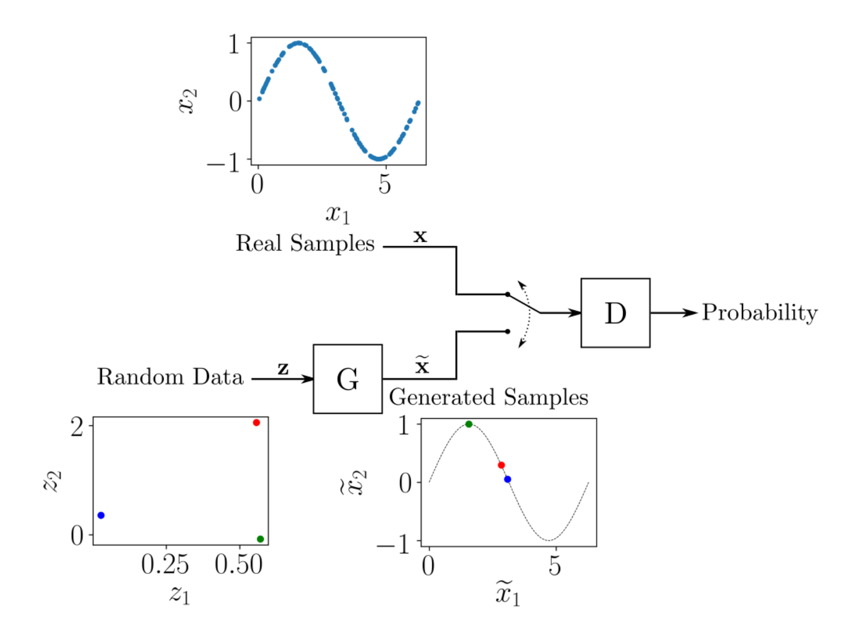
ويتمثل دور المولد في تقدير التوزيع الاحتمالي للعينات الحقيقية من أجل توفير العينات المولدة التي تشبه البيانات الحقيقية. ويتم تدريب المُميِّز بدوره على تقدير احتمالية أن تأتي عينة معينة من البيانات الحقيقية بدلاً من أن يقدمها المولد.

تسمى هذه الهياكل شبكات الخصومة التوليدية generative adversarial networks لأن المولد والمميز مدربان على التنافس مع بعضهما البعض: يحاول المولد أن يتحسن في خداع المُميِّز، بينما يحاول المُميِّز أن يتحسن في تحديد العينات المولدة.

لفهم كيفية عمل تدريب GAN، فكر في مثال لعبة مع مجموعة بيانات مكونة من عينات ثنائية الأبعاد (x_1, x_2) ، مع x_1 في الفترة من 0 إلى 2π و$x_2 = \sin(x_1)$، كما هو موضح في الشكل التالي:



كما ترون، تتكون مجموعة البيانات هذه من نقاط (x_1, x_2) تقع فوق منحنى جيبي sine curve، ولها توزيع خاص جداً. يظهر الشكل التالي الهيكل العام لشبكة GAN لإنشاء أزواج ($\tilde{x}_1$, $\tilde{x}_2$) تشبه عينات مجموعة البيانات:



يتم تغذية المولد G ببيانات عشوائية من الفضاء الكامن latent space، ودوره هو توليد بيانات تشبه العينات الحقيقية.في هذا المثال، لديك مساحة كامنة ثنائية الأبعاد، بحيث يتم تغذية المولد بأزواج عشوائية (z_1, z_2) ويطلب منه تحويلها بحيث تشبه العينات الحقيقية.

يمكن أن يكون هيكل الشبكة العصبية G تعسفيًا، مما يسمح لك باستخدام الشبكات العصبية كشبكة بيرسيبترون متعددة الطبقات multilayer perceptron (MLP)، أو شبكة عصبية تلافيفية

convolutional neural network (CNN)، أو أي معمارية أخرى طالما أن أبعاد المدخلات والمخرجات تتطابق مع الأبعاد الفضاء الكامن والبيانات الحقيقية.

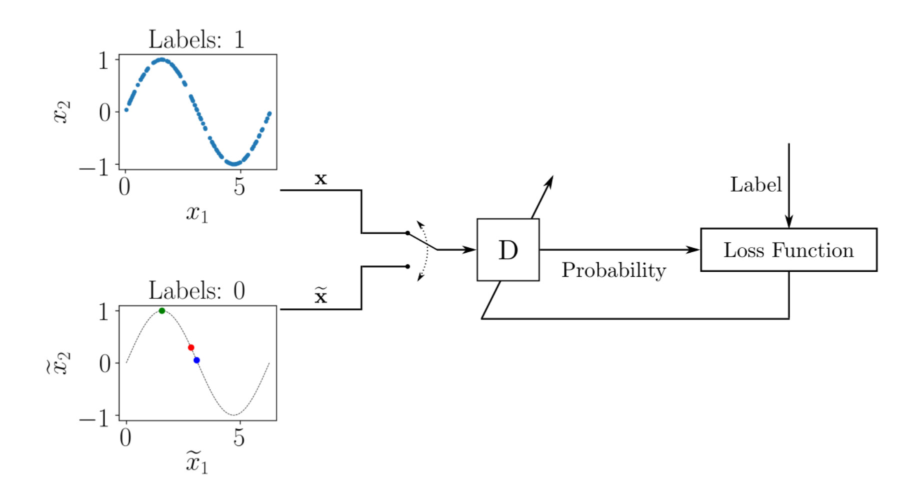
يتم تغذية المُميِّز D إما بعينات حقيقية من مجموعة بيانات التدريب أو العينات المُولدة المقدمة من G. ويتمثل دوره في تقدير احتمالية انتماء المدخلات إلى مجموعة البيانات الحقيقية. يتم تنفيذ التدريب بحيث يكون مخرج D هو 1 عندما يتم تغذيته بعينة حقيقية و0 عندما يتم تغذيته بعينة تم إنشاؤها.

كما هو الحال مع G، يمكنك اختيار بنية شبكة عصبية عشوائية لـ D طالما أنها تحترم أبعاد الإدخال والإخراج الضرورية.في هذا المثال، يكون الإدخال ثنائي الأبعاد. بالنسبة للمميز الثنائي، قد يكون الإخراج عددًا يتراوح من 0 إلى 1.

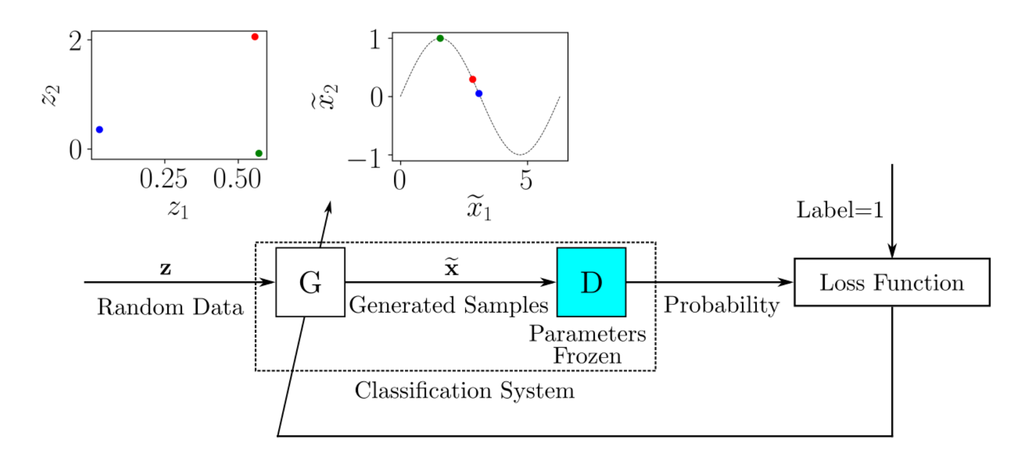
تتكون عملية تدريب GAN من لعبة minimax للاعبين حيث يتم تكييف D لتقليل خطأ التمييز بين العينات الحقيقية والمولدة، ويتم تكييف G لزيادة احتمالية ارتكاب D للخطأ.

على الرغم من عدم تصنيف مجموعة البيانات التي تحتوي على البيانات الحقيقية، إلا أن عمليات التدريب الخاصة بـ D وG يتم تنفيذها بطريقة خاضعة للإشراف.في كل خطوة من التدريب، يتم تحديث معلمات D وG.في الواقع،في اقتراح GAN الأصلي، يتم تحديث معلمات D لـ k من المرات، بينما يتم تحديث معلمات G مرة واحدة فقط لكل خطوة تدريب. ومع ذلك، لجعل التدريب أسهل، يمكنك اعتبار k يساوي 1.

لتدريب D،في كل تكرار، يمكنك تسمية بعض العينات الحقيقية المأخوذة من بيانات التدريب بالرقم 1 وبعض العينات التي تم إنشاؤها المقدمة من G بالرقم 0. وبهذه الطريقة، يمكنك استخدام إطار تدريب تقليدي خاضع للإشراف لتحديث معلمات D من أجل تقليل دالة الخطأ، كما هو موضح في المخطط التالي:



بالنسبة لكل دفعة من بيانات التدريب التي تحتوي على عينات حقيقية ومولدة، يمكنك تحديث معلمات D لتقليل دالة الخطأ. بعد تحديث معلمات D، يمكنك تدريب G لإنتاج عينات تم إنشاؤها بشكل أفضل. يتم توصيل مخرج G بـ D، الذي يتم الاحتفاظ بمعلماته مجمدة، كما هو موضح هنا:



يمكنك أن تتخيل النظام المكون من G و D كنظام تصنيف واحد يستقبل عينات عشوائية كمدخلات ويخرج التصنيف، وهو ما يمكن تفسيره في هذه الحالة على أنه احتمال.

عندما يقوم G بعمل جيد بما فيه الكفاية لخداع D، يجب أن يكون احتمال الناتج قريبًا من 1. يمكنك أيضًا استخدام إطار تدريب تقليدي خاضع للإشراف هنا: سيتم توفير مجموعة البيانات لتدريب نظام التصنيف المكون من G وD من خلال عينات إدخال عشوائية، وستكون التسمية المرتبطة بكل عينة إدخال هي 1.

أثناء التدريب، مع تحديث معلمات D وG، من المتوقع أن تكون العينات التي تم إنشاؤها المقدمة بواسطة G أكثر تشابهًا مع البيانات الحقيقية، وسيواجه D المزيد من المشاكل في التمييز بين البيانات الحقيقية والمولدة.

الآن بعد أن عرفت كيفية عمل شبكات GAN، فأنت جاهز لتنفيذ شبكتك الخاصة باستخدام PyTorch.

شبكة GAN الأولى لديك

كتجربة أولى مع شبكات الخصومة التوليدية، ستقوم بتنفيذ المثال الموضح في القسم السابق.

لتشغيل المثال، ستستخدم مكتبة PyTorch، والتي يمكنك تثبيتها باستخدام توزيع Anaconda Python وحزمة conda ونظام إدارة البيئة. لمعرفة المزيد حول Anaconda وconda، راجع البرنامج التعليمي حول إعداد Python للتعلم الآلي على Windows.

للبدء، أنشئ بيئة conda وقم بتنشيطها:

```
$ conda create --name gan
$ conda activate gan
```

بعد تنشيط بيئة conda، ستظهر المطالبة باسمها، gan. ثم يمكنك تثبيت الحزم اللازمة داخل البيئة:

```
$ conda install -c pytorch pytorch=1.4.0
$ conda install matplotlib jupyter
```

نظرًا لأن PyTorch عبارة عن إطار عمل تم تطويره بشكل نشط للغاية، فقد تتغير واجهة برمجة التطبيقات (API)في الإصدارات الجديدة. للتأكد من تشغيل كود المثال، قم بتثبيت الإصدار المحدد 1.4.0.

إلى جانب PyTorch، ستستخدم Matplotlib للعمل مع الرسومات البيانية و Jupyter Notebook لتشغيل التعليمات البرمجيةفي بيئة تفاعلية. القيام بذلك ليس إلزاميًا، ولكنه يسهل العمل على مشاريع التعلم الآلي.

لتجديد المعلومات حول العمل مع Matplotlib وJupyter Notebooks، قم بإلقاء نظرة على Python Plotting With Matplotlib (الدليل) وJupyter Notebook: مقدمة.

قبل فتح Jupyter Notebook، تحتاج إلى تسجيل بيئة conda gan حتى تتمكن من إنشاء Notebooks باستخدامها كنواة. للقيام بذلك، بعد تفعيل بيئة gan، قم بتشغيل الأمر التالي:

```
$ python -m ipykernel install --user --name gan
```

يمكنك الآن فتح Jupyter Notebook عن طريق تشغيل jupyter Notebook. قم بإنشاء نوتبوك جديد بالنقر فوق جديد ثم تحديد gan.

داخل Notebook، ابدأ باستيراد المكتبات الضرورية:

```
import torch
from torch import nn

import math
import matplotlib.pyplot as plt
```

هنا، يمكنك استيراد مكتبة PyTorch باستخدام torch. يمكنك أيضًا استيراد nn فقط لتتمكن من إعداد الشبكات العصبية بطريقة أقل تفصيلاً. ثم تقوم باستيراد math للحصول على قيمة ثابت pi، وتقوم باستيراد أدوات رسم Matplotlib كـ plt كالمعتاد.

من الممارسات الجيدة إعداد بذرة مولد عشوائي random generator seed بحيث يمكن تكرار التجربة بشكل مماثل على أي جهاز. للقيام بذلك في PyTorch، قم بتشغيل التعليمات البرمجية التالية:

```
torch.manual_seed(111)
```

إعداد بيانات التدريب

تتكون بيانات التدريب من أزواج ((x_1, x_2)) بحيث تتكون x_2 من قيمة جيب x_1 لـ x_1 في الفترة من 0 إلى 2π. يمكنك تنفيذه على النحو التالي:

```
train_data = torch.zeros((train_data_length, 2))
 3train_data[:, 0] = 2 * math.pi *
torch.rand(train_data_length)
 4train_data[:, 1] = torch.sin(train_data[:, 0])
 5train_labels = torch.zeros(train_data_length)
 6train_set = [
 7     (train_data[i], train_labels[i]) for i in
range(train_data_length)
 8]
```

هنا، تقوم بتكوين مجموعة تدريب مكونة من 1024 زوجًا (x_1, x_2).في السطر 2، قمت بتهيئة Train_data، وهو موتر بأبعاد 1024 صفًا وعمودين، تحتوي جميعها على أصفار. الموتر tensor عبارة عن مصفوفة متعددة الأبعاد تشبه مصفوفة NumPy.

في السطر 3، يمكنك استخدام العمود الأول من Train_data لتخزين القيم العشوائية في الفترة من 0 إلى 2π. ثم،في السطر 4، يمكنك حساب العمود الثاني من الموتر باعتباره جيب العمود الأول.

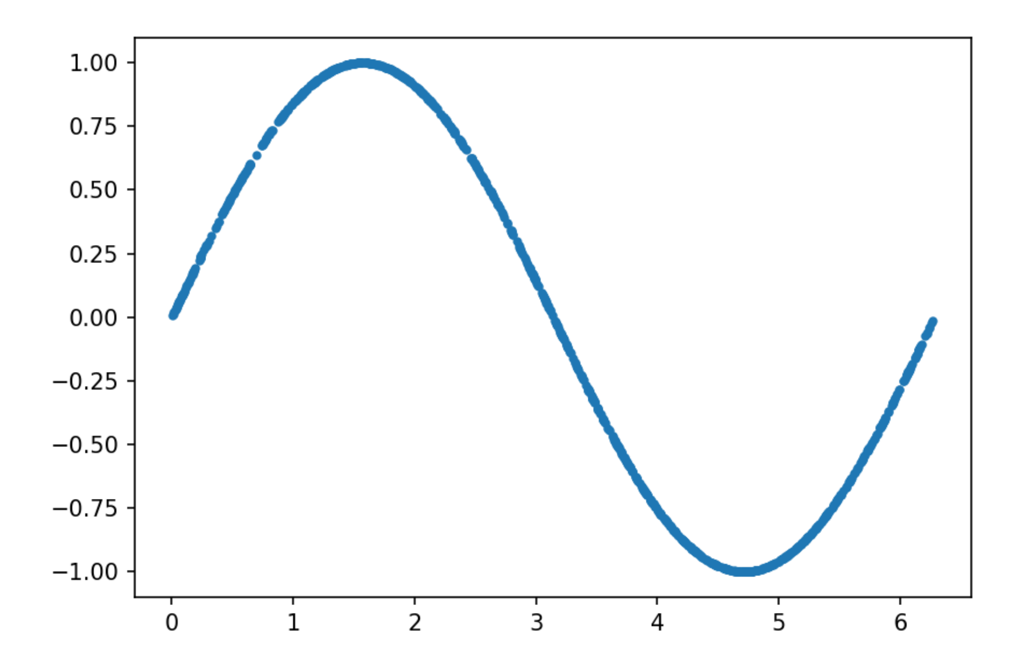
بعد ذلك، ستحتاج إلى موتر من التسميات labels، التي يطلبها مُحمل بيانات PyTorch. نظرًا لأن شبكات GAN تستخدم تقنيات التعلم غير الخاضعة للأشراف، فيمكن أن تكون التسميات أي شيء. لن يتم استخدامها بعد كل شيء.

في السطر 5، قمت بإنشاء Train_labels، وهو موتر مليء بالأصفار. أخيرًا،في الأسطر من 6 إلى 8، يمكنك إنشاء Train_set كقائمة من المجموعات، مع تمثيل كل صف من Train_data وtrain_labels في كل صف كما هو متوقع بواسطة أداة تحميل بيانات data loader من PyTorch.

يمكنك فحص بيانات التدريب من خلال رسم كل نقطة (x_1, x_2):

```
plt.plot(train_data[:, 0], train_data[:, 1], ".")
```

يجب أن يكون الإخراج مشابهًا للشكل التالي:



باستخدام Train_set، يمكنك إنشاء أداة تحميل بيانات PyTorch:

```
batch_size = 32
train_loader = torch.utils.data.DataLoader(
    train_set, batch_size=batch_size, shuffle=True
)
```

هنا، يمكنك إنشاء أداة تحميل بيانات تسمى Train_loader، والتي ستقوم بخلط البيانات من Train_set وإرجاع دفعات batches مكونة من 32 عينة ستستخدمها لتدريب الشبكات العصبية.

بعد إعداد بيانات التدريب، تحتاج إلى إنشاء الشبكات العصبية للمميز والمولد الذي سيشكل شبكة GAN. في القسم التالي، عليك تنفيذ المميز.

تنفيذ المميز

في PyTorch، يتم تمثيل نماذج الشبكة العصبية بواسطة فئات ترث من nn.Module، لذلك سيتعين عليك تحديد فئة لإنشاء المميز. لمزيد من المعلومات حول تعريف الفئات، قم بإلقاء نظرة على البرمجة كائنية التوجه (OOP) في Python 3.

المميز هو نموذج ذو مدخلات ثنائية الأبعاد ومخرجات أحادية البعد. ستتلقى عينة من البيانات الحقيقية أو من المولد وستوفر احتمالية أن تنتمي العينة إلى بيانات التدريب الحقيقية. يوضح الكود أدناه كيفية إنشاء المميز:

```
class Discriminator(nn.Module):
```

```
2    def __init__(self):
3        super().__init__()
4        self.model = nn.Sequential(
5            nn.Linear(2, 256),
6            nn.ReLU(),
7            nn.Dropout(0.3),
8            nn.Linear(256, 128),
9            nn.ReLU(),
10           nn.Dropout(0.3),
11           nn.Linear(128, 64),
12           nn.ReLU(),
13           nn.Dropout(0.3),
14           nn.Linear(64, 1),
15           nn.Sigmoid(),
16       )
17
18   def forward(self, x):
19       output = self.model(x)
20       return output
```

يمكنك استخدام ()__init__ . لبناء النموذج. أولاً، تحتاج إلى استدعاء ()__init__.()super لتشغيل ()__init__ . من nn.Module. المميز التي تستخدمها هي شبكة عصبية MLP محددة بطريقة تسلسلية باستخدام ()nn.Sequential. لديها الخصائص التالية:

- **السطران 5 و 6**: الإدخال ثنائي الأبعاد، وتتكون الطبقة المخفية الأولى من 256 خلية عصبية مع تنشيط ReLU.
- **الأسطر 8 و9 و11 و12**: تتكون الطبقتان المخفية الثانية والثالثة من 128 و64 خلية عصبية، على التوالي، مع تنشيط ReLU.
- **السطران 14 و15**: يتكون الناتج من خلية عصبية واحدة ذات تنشيط سيني sigmoidal لتمثيل الاحتمال.
- **الأسطر 7 و10 و13**: بعد الطبقات المخفية الأولى والثانية والثالثة، يمكنك استخدام التسرب dropout لتجنب الضبط الزائد overfitting.

وأخيرًا، يمكنك استخدام ()forward. لوصف كيفية حساب مخرجات النموذج. هنا، تمثل x مدخلات النموذج، وهو موتر ثنائي الأبعاد.في هذا التنفيذ، يتم الحصول على الإخراج عن طريق تغذية المدخلات x للنموذج الذي حددته دون أي معالجة أخرى.

بعد الإعلان عن فئة المميز، يجب عليك إنشاء مثيل لكائن Discriminator:

```
discriminator = Discriminator()
```

يمثل التمييز مثيلاً للشبكة العصبية التي حددتها وجاهزة للتدريب. ومع ذلك، قبل تنفيذ حلقة التدريب، تحتاج شبكة GAN الخاصة بك أيضًا إلى مولد. ستقوم بتنفيذ واحدفي القسم التالي.

تنفيذ المولد

في شبكات الخصومة التوليدية GANs، المولد هو النموذج الذي يأخذ عينات من الفضاء الكامن كمدخلات له ويولد بيانات تشبه البيانات الموجودةفي مجموعة التدريب.في هذه الحالة، هو نموذج ذو مدخل ثنائي الأبعاد، والذي سيتلقى نقاط عشوائية (z_1 ،z_2)، ومخرجات ثنائية الأبعاد يجب أن توفر ($\tilde{x}_1$ ،$\tilde{x}_2$) نقاط تشبه تلك الموجودةفي بيانات التدريب.

التنفيذ مشابه لما فعلته بالنسبة للمميز. أولاً، عليك إنشاء فئة Generator ترث من nn.Module، وتحديد معمارية الشبكة العصبية، ثم تحتاج إلى إنشاء كائن Generator:

```
class Generator(nn.Module):
 2    def __init__(self):
 3        super().__init__()
 4        self.model = nn.Sequential(
 5            nn.Linear(2, 16),
 6            nn.ReLU(),
 7            nn.Linear(16, 32),
 8            nn.ReLU(),
 9            nn.Linear(32, 2),
10        )
11
12    def forward(self, x):
13        output = self.model(x)
14        return output
15
16generator = Generator()
```

هنا، يمثل generator الشبكة العصبية للمولد. وهي تتألف من طبقتين مخفيتين تحتويان على 16 و32 خلية عصبية، كلاهما مع تنشيط ReLU، وطبقة تنشيط خطية تحتوي على خليتين عصبيتين في الإخراج. بهذه الطريقة، سيتكون الإخراج من متجه بعنصرين يمكن أن يكونا أي قيمة تتراوح من اللانهاية السالبة إلى اللانهاية، والتي ستمثل ($\tilde{x}_1$ ،$\tilde{x}_2$).

الآن بعد أن حددت نماذج المُميِّز والمولد، أنت جاهز لأداء التدريب!

تدريب النماذج

قبل تدريب النماذج، تحتاج إلى إعداد بعض المعلمات لاستخدامها أثناء التدريب:

```
lr = 0.001
num_epochs = 300
loss_function = nn.BCELoss()
```

هنا قمت بإعداد المعلمات التالية:

- **يضبط السطر 1** معدل التعلم (lr)، الذي ستستخدمه لتكييف أوزان الشبكة.

- **يحدد السطر 2** عدد الفترات (num_epochs)، والذي يحدد عدد مرات تكرار التدريب باستخدام مجموعة التدريب بأكملها التي سيتم تنفيذها.
- **يقوم السطر 3** بتعيين المتغير loss_function إلى دالة الإنتروبيا المتقاطعة الثنائية binary cross-entropy function، وهي دالة الخطأ التي ستستخدمها لتدريب النماذج.

تعد دالة الإنتروبيا المتقاطعة الثنائية بمثابة دالة خطأ مناسبة لتدريب المُميِّز لأنها تعتبر مهمة تصنيف ثنائية. كما أنها مناسبة لتدريب المولد لأنه يغذي مخرجاته إلى المميز، والتي توفر مخرجات ثنائية يمكن ملاحظتها.

تطبق PyTorch قواعد مختلفة لتحديث الوزن لتدريب النموذج في torch.optim. ستستخدم خوارزمية Adam لتدريب نماذج المميز والمولد. لإنشاء أدوات تحسين الأداء باستخدام torch.optim، قم بتشغيل الأسطر التالية:

```
optimizer_discriminator =
torch.optim.Adam(discriminator.parameters(), lr=lr)
optimizer_generator =
torch.optim.Adam(generator.parameters(), lr=lr)
```

أخيرًا، تحتاج إلى تنفيذ حلقة تدريب يتم فيها تغذية النماذج بعينات التدريب، ويتم تحديث أوزانها لتقليل دالة الخطأ:

```
 1 for epoch in range(num_epochs):
 2     for n, (real_samples, _) in enumerate(train_loader):
 3         # Data for training the discriminator
 4         real_samples_labels = torch.ones((batch_size, 1))
 5         latent_space_samples = torch.randn((batch_size,
2))
 6         generated_samples =
generator(latent_space_samples)
 7         generated_samples_labels =
torch.zeros((batch_size, 1))
 8         all_samples = torch.cat((real_samples,
generated_samples))
 9         all_samples_labels = torch.cat(
10             (real_samples_labels,
generated_samples_labels)
11         )
12
13         # Training the discriminator
14         discriminator.zero_grad()
15         output_discriminator = discriminator(all_samples)
16         loss_discriminator = loss_function(
17             output_discriminator, all_samples_labels)
18         loss_discriminator.backward()
19         optimizer_discriminator.step()
```

```
20
21         # Data for training the generator
22         latent_space_samples = torch.randn((batch_size,
2))
23
24         # Training the generator
25         generator.zero_grad()
26         generated_samples =
generator(latent_space_samples)
27         output_discriminator_generated =
discriminator(generated_samples)
28         loss_generator = loss_function(
29             output_discriminator_generated,
real_samples_labels
30         )
31         loss_generator.backward()
32         optimizer_generator.step()
33
34         # Show loss
35         if epoch % 10 == 0 and n == batch_size - 1:
36             print(f"Epoch: {epoch} Loss D.:
{loss_discriminator}")
37             print(f"Epoch: {epoch} Loss G.:
{loss_generator}")
```

بالنسبة لشبكات GAN، يمكنك تحديث معلمات المُميِّز والمولدفي كل تكرار تدريب. كما هو الحال عمومًافي جميع الشبكات العصبية، تتكون عملية التدريب من حلقتين، واحدة لفترات التدريب والأخرى للدفعات لكل فترة. داخل الحلقة الداخلية، تبدأفي إعداد البيانات لتدريب المُميِّز:

- **السطر 2:** تحصل على العينات الحقيقية للدفعة الحالية من أداة تحميل البيانات وتعيينها إلى real_samples. لاحظ أن البعد الأول للموتر يحتوي على عدد من العناصر يساوي Batt_size. هذه هي الطريقة القياسية لتنظيم البياناتفي PyTorch، حيث يمثل كل سطر من الموتر عينة واحدة من الدفعة.
- **السطر 4:** يمكنك استخدام ()torch.ones لإنشاء تسميات بالقيمة 1 للعينات الحقيقية، ثم تقوم بتعيين التسميات إلى real_samples_labels.
- **السطران 5 و6:** يمكنك إنشاء العينات التي تم إنشاؤها عن طريق تخزين البيانات العشوائية في latent_space_samples، والتي تقوم بعد ذلك بادخالها للمولد للحصول على العينات التي تم إنشاؤها.
- **السطر 7:** يمكنك استخدام ()torch.zeros لتعيين القيمة 0 للتسميات الخاصة بالعينات التي تم إنشاؤها، ثم تقوم بتخزين التسمياتفي generator_samples_labels.
- **الأسطر من 8 إلى 11:** تقوم بتسلسل العينات والتسميات الحقيقية والمولدة وتخزينهافي all_samples وall_samples_labels، والتي ستستخدمها لتدريب المُميِّز.

بعد ذلك،في السطور من 14 إلى 19، تقوم بتدريب المُميِّز:

- **السطر 14:**في PyTorch، من الضروري مسح التدرجاتفي كل خطوة تدريب لتجنب تراكمها. يمكنك القيام بذلك باستخدام zero_grad()..
- **السطر 15:** تقوم بحساب مخرجات المُميز باستخدام بيانات التدريب الموجودةفي all_samples.
- **السطران 16 و17:** يمكنك حساب دالة الخطأ باستخدام الإخراج من النموذج في output_discriminator والتسمياتفي all_samples_labels.
- **السطر 18:** تقوم بحساب التدرجات لتحديث الأوزان باستخدام loss_discriminator.backward().
- **السطر 19:** يمكنك تحديث أوزان المميز عن طريق استدعاء optimizer_discriminator.step().

بعد ذلك،في السطر 22، تقوم بإعداد البيانات لتدريب المولد. يمكنك تخزين البيانات العشوائيةفي latent_space_samples، مع عدد من الأسطر يساوي Batch_size. يمكنك استخدام عمودين نظرًا لأنك تقدم بيانات ثنائية الأبعاد كمدخلات للمولد.

تقوم بتدريب المولدفي الأسطر من 25 إلى 32:

- **السطر 25:** يمكنك مسح التدرجات باستخدام zero_grad().
- **السطر 26:** تقوم بتغذية المولد باستخدام latent_space_samplesوتخزين مخرجاتهفي generated_samples.
- **السطر 27:** تقوم بتغذية مخرجات المولدفي المميز وتخزين مخرجاتهافي مخرجات discriminator_generated، والتي ستستخدمها كمخرجات النموذج بأكمله.
- **الأسطر من 28 إلى 30:** يمكنك حساب دالة الخطأ باستخدام مخرجات نظام التصنيف المخزنفي input_discriminator_generated والتسميات الموجودةفي real_samples_labels، والتي تساوي جميعها 1.
- **السطر 31 و 32:** تقوم بحساب التدرجات وتحديث أوزان المولد. تذكر أنه عندما قمت بتدريب المولد، أبقيت أوزان المميز مجمدة منذ أن قمت بإنشاء optimizer_generator مع وسيطه الأول الذي يساوي generator.parameters().

أخيرًا،في الأسطر من 35 إلى 37، يمكنك عرض قيم دالة خطأ المميز والمولدفي نهاية كل عشرة فترات.

نظرًا لأن النماذج المستخدمةفي هذا المثال تحتوي على معلمات قليلة، فسيتم إكمال التدريب خلال دقائق قليلة.في القسم التالي، ستستخدم شبكة GAN المدربة لإنشاء بعض العينات.

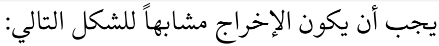
التحقق من العينات التي تم إنشاؤها بواسطة GAN

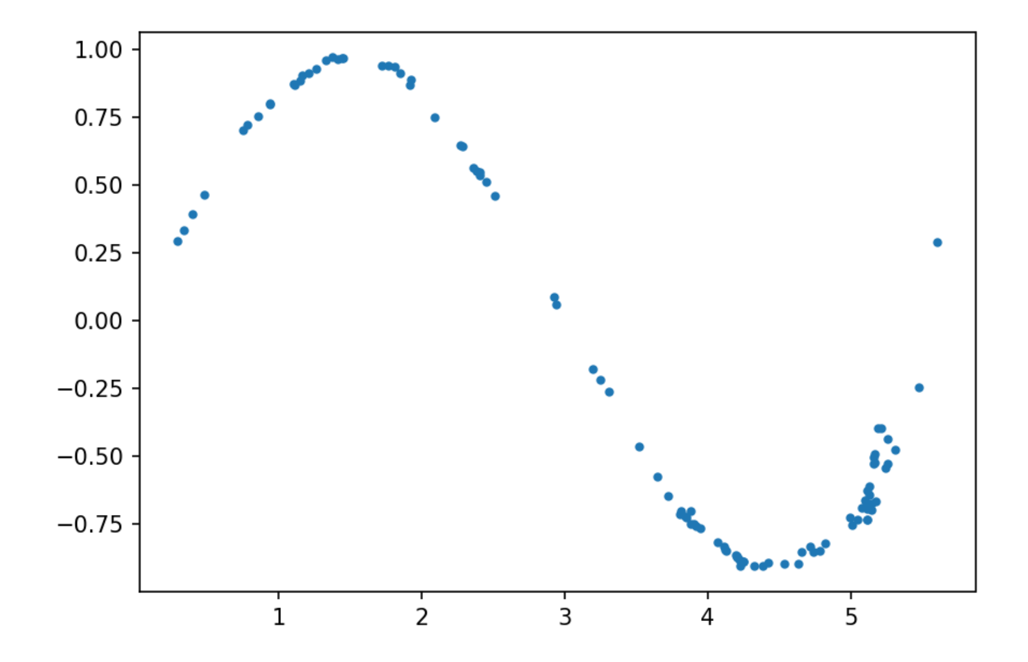
تم تصميم شبكات الخصومة التوليدية لتوليد البيانات. لذلك، بعد الانتهاء من عملية التدريب، يمكنك الحصول على بعض العينات العشوائية من الفضاء الكامن وتغذيتها للمولد للحصول على بعض العينات المولدة:

```
latent_space_samples = torch.randn(100, 2)
generated_samples = generator(latent_space_samples)
```

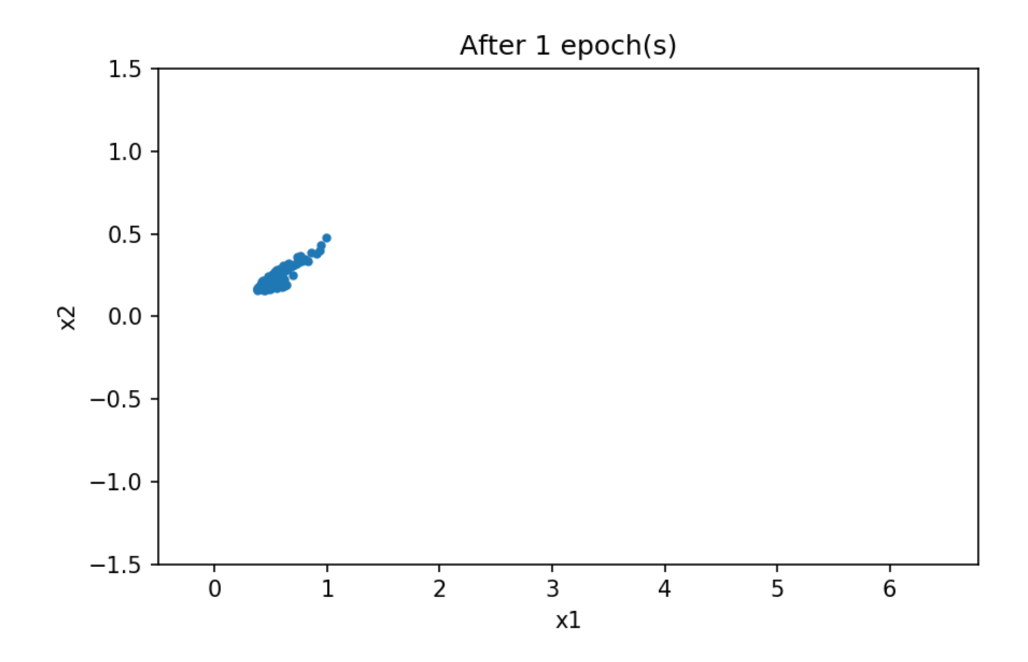
ثم يمكنك رسم العينات التي تم إنشاؤها والتحقق مما إذا كانت تشبه بيانات التدريب. قبل رسم بيانات generated_samples، ستحتاج إلى استخدام()detach. لإرجاع موتر من الرسم البياني الحسابي لـ PyTorch، والذي ستستخدمه بعد ذلك لحساب التدرجات:

```
generated_samples = generated_samples.detach()
plt.plot(generated_samples[:, 0], generated_samples[:, 1],
".")
```


يجب أن يكون الإخراج مشابهاً للشكل التالي:



يمكنك أن ترى أن توزيع البيانات التي تم إنشاؤها يشبه توزيع البيانات الحقيقية. باستخدام موتر عينات الفضاء الكامنة الثابتة وادخاله إلى المولدفي نهاية كل فترة أثناء عملية التدريب، يمكنك تصور تطور التدريب:



لاحظ أنهفي بداية عملية التدريب، يختلف توزيع البيانات الناتجة كثيرًا عن البيانات الحقيقية. ومع ذلك، مع تقدم التدريب، يتعلم المولد التوزيع الحقيقي للبيانات.

الآن بعد أن انتهيت من التنفيذ الأول لشبكة الخصومة التوليدية، ستنتقل إلى تطبيق أكثر عملية باستخدام الصور.

مولد أرقام مكتوبة بخط اليد مع GAN

يمكن لشبكات الخصومة التوليدية أيضًا إنشاء عينات عالية الأبعاد مثل الصور.في هذا المثال، ستستخدم GAN لإنشاء صور للأرقام المكتوبة بخط اليد. لتحقيق ذلك، ستقوم بتدريب النماذج باستخدام مجموعة بيانات MNIST المكونة من أرقام مكتوبة بخط اليد، والتي تم تضمينهافي حزمة torchvision.

للبدء، تحتاج إلى تثبيت torchvisionفي بيئة gan conda المنشَّطة:

```
$ conda install -c pytorch torchvision=0.5.0
```

مرة أخرى، أنت تستخدم إصدارًا محددًا من torchvision لضمان تشغيل كود المثال، تمامًا كما فعلت مع pytorch. بعد إعداد البيئة، يمكنك البدء في تنفيذ النماذج في Jupyter Notebook. افتحه وقم بإنشاء Notebook جديد بالنقر فوق جديد ثم تحديد gan.

كما في المثال السابق، تبدأ باستيراد المكتبات الضرورية:

```
import torch
from torch import nn

import math
import matplotlib.pyplot as plt
import torchvision
import torchvision.transforms as transforms
```

إلى جانب المكتبات التي استوردتها من قبل، ستحتاج إلى torchvision والتحويلات للحصول على بيانات التدريب وإجراء تحويلات الصور.

مرة أخرى، قم بإعداد بذرة المولد العشوائي random generator seed لتتمكن من تكرار التجربة:

```
torch.manual_seed(111)
```

نظرًا لأن هذا المثال يستخدم صورًا في مجموعة التدريب، فيجب أن تكون النماذج أكثر تعقيدًا، مع عدد أكبر من المعلمات. وهذا يجعل عملية التدريب أبطأ، وتستغرق حوالي دقيقتين لكل فترة عند التشغيل على وحدة المعالجة المركزية CPU. ستحتاج إلى حوالي خمسين فترة للحصول على نتيجة ذات صلة، وبالتالي فإن إجمالي وقت التدريب عند استخدام CPU يبلغ حوالي مائة دقيقة.

لتقليل وقت التدريب، يمكنك استخدام وحدة معالجة الرسوميات (GPU) لتدريب النموذج إذا كان لديك واحدة متاحة. ومع ذلك، ستحتاج إلى نقل الموترات والنماذج يدويًا إلى GPU لاستخدامها في عملية التدريب.

يمكنك التأكد من تشغيل التعليمات البرمجية الخاصة بك في أي من الإعدادين عن طريق إنشاء كائن device يشير إما إلى وحدة المعالجة المركزية (CPU)، أو إلى وحدة معالجة الرسومات (GPU)، إذا كان متاحًا:

```
device = ""
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")
```

لاحقًا، ستستخدم هذا device لتعيين المكان الذي يجب إنشاء الموترات والنماذج فيه، باستخدام وحدة معالجة الرسومات (GPU) إذا كانت متوفرة.

الآن بعد أن تم تعيين البيئة الأساسية، يمكنك إعداد بيانات التدريب.

إعداد بيانات التدريب

تتكون مجموعة بيانات MNIST من صور ذات تدرج رمادي مقاس 28 × 28 بكسل لأرقام مكتوبة بخط اليد من 0 إلى 9. لاستخدامها مع PyTorch، ستحتاج إلى إجراء بعض التحويلات. لذلك، يمكنك تعريف transform، وهي دالة سيتم استخدامها عند تحميل البيانات:

```
transform = transforms.Compose(
    [transforms.ToTensor(), transforms.Normalize((0.5,),
(0.5,))]
)
```

تتكون الدالة من جزأين:

1. يقوم ()transforms.ToTensor بتحويل البيانات إلى موتر PyTorch.
2. تقوم الدالة ()Transforms.Normalize بتحويل نطاق معاملات الموتر.

تتراوح المعاملات الأصلية المقدمة بواسطة ()transforms.ToTensor من 0 إلى 1، وبما أن خلفيات الصورة سوداء، فإن معظم المعاملات تساوي 0 عندما يتم تمثيلها باستخدام هذا النطاق.

يقوم الدالة ()Transforms.Normalize بتغيير نطاق المعاملات من ـ1 إلى 1 عن طريق طرح 0.5 من المعاملات الأصلية وتقسيم النتيجة على 0.5. وبهذا التحويل، يتم تقليل عدد العناصر التي تساوي 0في العينات المدخلة بشكل كبير، مما يساعدفي تدريب النماذج.

وسيطات ()transforms.Normalize عبارة عن صفين، (M_1، ....، M_n) و (S_1، ....، S_n)، حيث يمثل n عدد قنوات الصور. تحتوي الصور ذات التدرج الرمادي مثل تلك الموجودةفي مجموعة بيانات MNIST على قناة واحدة فقط، وبالتالي فإن المجموعات لها قيمة واحدة فقط. بعد ذلك، لكل قناة i من الصورة، تقوم الدالة ()transforms.Normalize بطرح M_i من المعاملات وتقسيم النتيجة على S_i.

يمكنك الآن تحميل بيانات التدريب باستخدام torchvision.datasets.MNIST وإجراء التحويلات باستخدام transform:

```
train_set = torchvision.datasets.MNIST(
    root=".", train=True, download=True, transform=transform
)
```

تضمن الوسيط download=True أنهفي المرة الأولى التي تقوم فيها بتشغيل التعليمات البرمجية أعلاه، سيتم تنزيل مجموعة بيانات MNIST وتخزينهافي الدليل الحالي، كما هو موضح بواسطة الوسيط root.

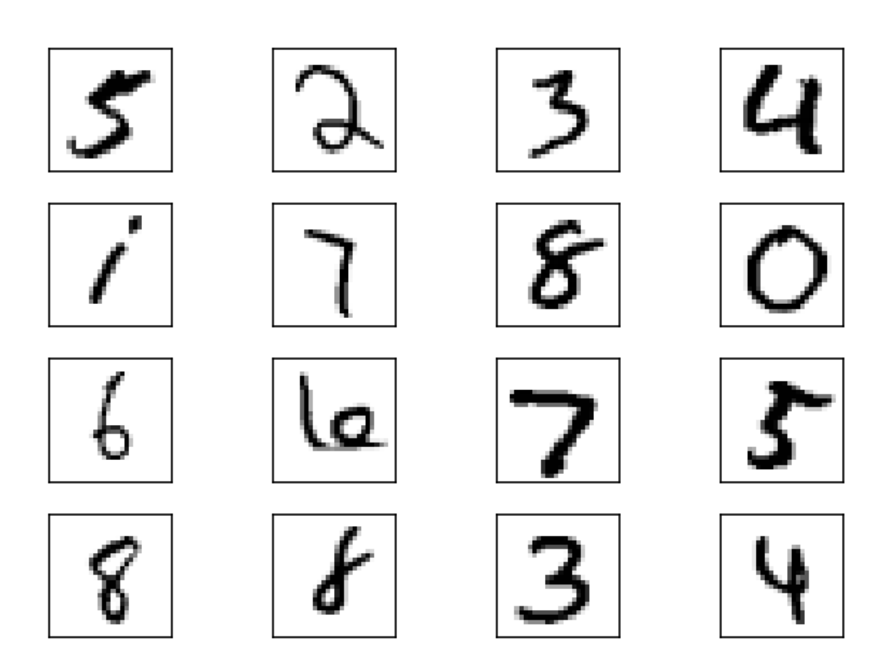
الآن بعد أن قمت بإنشاء Train_set، يمكنك إنشاء أداة تحميل البيانات data loader كما فعلت من قبل:

```
batch_size = 32
train_loader = torch.utils.data.DataLoader(
    train_set, batch_size=batch_size, shuffle=True
)
```

يمكنك استخدام Matplotlib لرسم بعض عينات بيانات التدريب. لتحسين التصور، يمكنك استخدام cmap=gray_r لعكس خريطة الألوان ورسم الأرقام باللون الأسود على خلفية بيضاء:

```
real_samples, mnist_labels = next(iter(train_loader))
for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(real_samples[i].reshape(28, 28),
cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
```

يجب أن يكون الإخراج شيئًا مشابهًا لما يلي:



كما ترون، هناك أرقام ذات أنماط الكتابة اليدوية المختلفة. عندما تتعلم GAN توزيع البيانات، فإنها ستقوم أيضًا بإنشاء أرقام بأنماط كتابة يدوية مختلفة.

الآن بعد أن قمت بإعداد بيانات التدريب، يمكنك تنفيذ نماذج المميز والمولد.

تنفيذ المميز والمولد

في هذه الحالة، المُميِّز عبارة عن شبكة عصبية MLP تستقبل صورة بحجم 28 × 28 بكسل وتوفر احتمالية أن تنتمي الصورة إلى بيانات التدريب الحقيقية.

يمكنك تعريف النموذج بالكود التالي:

```
 1 class Discriminator(nn.Module):
 2   def __init__(self):
 3       super().__init__()
 4       self.model = nn.Sequential(
 5          nn.Linear(784, 1024),
 6          nn.ReLU(),
 7          nn.Dropout(0.3),
 8          nn.Linear(1024, 512),
 9          nn.ReLU(),
10          nn.Dropout(0.3),
11          nn.Linear(512, 256),
12          nn.ReLU(),
13          nn.Dropout(0.3),
14          nn.Linear(256, 1),
15          nn.Sigmoid(),
16       )
17
18   def forward(self, x):
19       x = x.view(x.size(0), 784)
20       output = self.model(x)
21       return output
```

لإدخال معاملات الصورةفي الشبكة العصبية MLP، يمكنك توجيهها بحيث تستقبل الشبكة العصبية متجهات ذات 784 معاملًا.

يحدث التوجيه vectorizationفي السطر الأول من()forward.، حيث يؤدي استدعاء ()x.view إلى تحويل شكل موتر الإدخال.في هذه الحالة، الشكل الأصلي للمدخل x هو 32 × 1 × 28 × 28، حيث 32 هو حجم الدفعة batch size التي قمت بإعدادها. بعد التحويل، يصبح شكل 784 × 32، حيث يمثل كل سطر معاملات صورة مجموعة التدريب.

لتشغيل نموذج المميز باستخدام وحدة معالجة الرسوميات GPU، يجب عليك إنشاء مثيل له وإرساله إلى GPU باستخدام ()to.. لاستخدام GPU عندما تكون متاحة، يمكنك إرسال النموذج إلى كائن device الذي قمت بإنشائه مسبقًا:

```
discriminator = Discriminator().to(device=device)
```

نظرًا لأن المولد سيقوم بإنشاء بيانات أكثر تعقيدًا، فمن الضروري زيادة أبعاد المدخلات من المساحة الكامنة.في هذه الحالة، سيتم تغذية المولد بمدخل ذي 100 بُعد وسيوفر مخرجات بمعاملات 784، والتي سيتم تنظيمهافي موتر 28 × 28 يمثل صورة.

إليك كود نموذج المولد الكامل:

```
 2 class Generator(nn.Module):
 2   def __init__(self):
 3       super().__init__()
 4       self.model = nn.Sequential(
 5           nn.Linear(100, 256),
 6           nn.ReLU(),
 7           nn.Linear(256, 512),
 8           nn.ReLU(),
 9           nn.Linear(512, 1024),
10           nn.ReLU(),
11           nn.Linear(1024, 784),
12           nn.Tanh(),
13       )
14
15   def forward(self, x):
16       output = self.model(x)
17       output = output.view(x.size(0), 1, 28, 28)
18       return output
19
20generator = Generator().to(device=device)
```

في السطر 12، يمكنك استخدام دالة الظل الزائدية Tanh() كتنشيط لطبقة الإخراج حيث يجب أن تكون معاملات الإخراج في الفاصل الزمني من ـ 1 إلى 1.في السطر 20، يمكنك إنشاء مثيل للمولد وإرساله إلى device لاستخدامه GPU إذا كان متاحًا.

الآن بعد أن حددت النماذج، ستقوم بتدريبهم باستخدام بيانات التدريب.

تدريب النماذج

لتدريب النماذج، تحتاج إلى تحديد معلمات التدريب والمحسنات كما فعلت في المثال السابق:

```
lr = 0.0001
num_epochs = 50
loss_function = nn.BCELoss()

optimizer_discriminator =
torch.optim.Adam(discriminator.parameters(), lr=lr)
optimizer_generator =
torch.optim.Adam(generator.parameters(), lr=lr)
```

للحصول على نتيجة أفضل تقوم بتقليل معدل التعلم عن المثال السابق. يمكنك أيضًا ضبط عدد الفترات على 50 لتقليل وقت التدريب.

حلقة التدريب مشابهة جدًا لتلك التي استخدمتهافي المثال السابق.في السطور المميزة، تقوم بإرسال بيانات التدريب إلى الجهاز لاستخدام وحدة معالجة الرسومات (GPU) إذا كانت متوفرة:

```
 1 for epoch in range(num_epochs):
 2     for n, (real_samples, mnist_labels) in
enumerate(train_loader):
 3         # Data for training the discriminator
 4         real_samples = real_samples.to(device=device)
 5         real_samples_labels = torch.ones((batch_size,
1)).to(
 6             device=device
 7         )
 8         latent_space_samples = torch.randn((batch_size,
100)).to(
 9             device=device
10         )
11         generated_samples =
generator(latent_space_samples)
12         generated_samples_labels =
torch.zeros((batch_size, 1)).to(
13             device=device
14         )
15         all_samples = torch.cat((real_samples,
generated_samples))
16         all_samples_labels = torch.cat(
17             (real_samples_labels,
generated_samples_labels)
18         )
19
```

```
20          # Training the discriminator
21          discriminator.zero_grad()
22          output_discriminator = discriminator(all_samples)
23          loss_discriminator = loss_function(
24              output_discriminator, all_samples_labels
25          )
26          loss_discriminator.backward()
27          optimizer_discriminator.step()
28
29          # Data for training the generator
30          latent_space_samples = torch.randn((batch_size,
100)).to(
31              device=device
32          )
33
34          # Training the generator
35          generator.zero_grad()
36          generated_samples =
generator(latent_space_samples)
37          output_discriminator_generated =
discriminator(generated_samples)
38          loss_generator = loss_function(
39              output_discriminator_generated,
real_samples_labels
40          )
41          loss_generator.backward()
42          optimizer_generator.step()
43
44          # Show loss
45          if n == batch_size - 1:
```

```
46              print(f"Epoch: {epoch} Loss D.:
{loss_discriminator}")
47              print(f"Epoch: {epoch} Loss G.:
{loss_generator}")
```

لا يلزم إرسال بعض الموترات إلى وحدة معالجة الرسوميات بشكل صريح مع device. هذا هو الحال بالنسبة للعينات التي تم إنشاؤهافي السطر 11، والتي سيتم إرسالها بالفعل إلى وحدة معالجة الرسومات المتاحة حيث تم إرسال latent_space_samples وgenerator إلى وحدة معالجة الرسومات مسبقاً.

نظرًا لأن هذا المثال يحتوي على نماذج أكثر تعقيدًا، فقد يستغرق التدريب وقتًا أطول قليلاً. وبعد الانتهاء، يمكنك التحقق من النتائج عن طريق إنشاء بعض نماذج الأرقام المكتوبة بخط اليد.

التحقق من العينات التي تم إنشاؤها بواسطة GAN

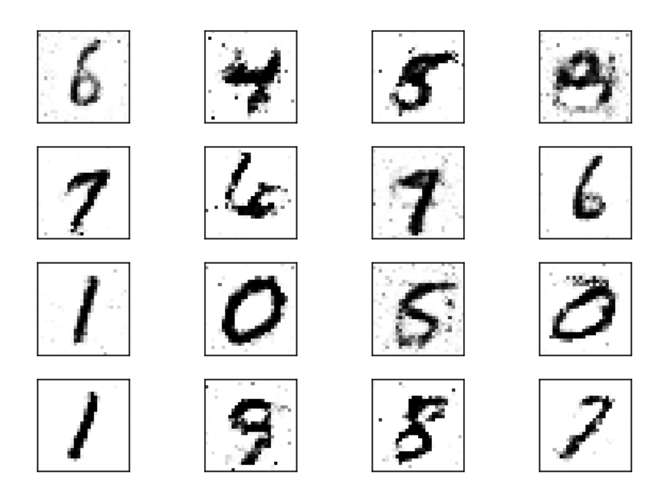
لتوليد أرقام مكتوبة بخط اليد، عليك أخذ بعض العينات العشوائية من الفضاء الكامن وتغذيتها للمولد:

```
latent_space_samples = torch.randn(batch_size,
100).to(device=device)
generated_samples = generator(latent_space_samples)
```

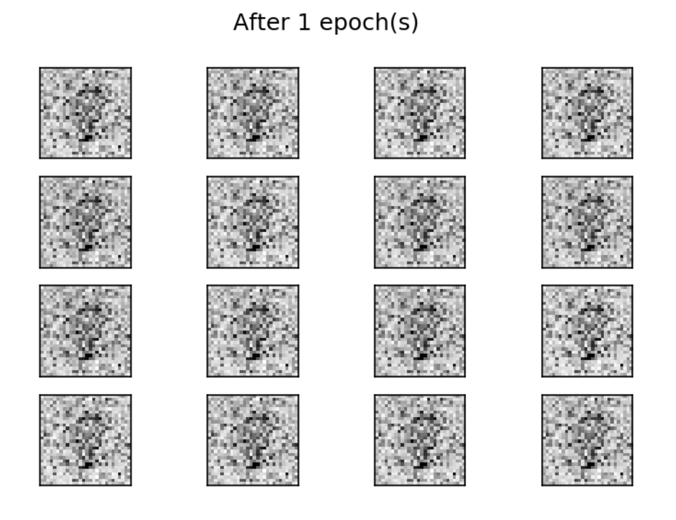
لرسم العينات التي تم إنشاؤها، تحتاج إلى نقل البيانات مرة أخرى إلى وحدة المعالجة المركزية CPU في حالة تشغيلها على وحدة معالجة الرسومات GPU. للقيام بذلك، يمكنك ببساطة الاتصال بـ .cpu(). كما فعلت سابقًا، تحتاج أيضًا إلى استدعاء .detach() قبل استخدام Matplotlib لرسم البيانات:

```
generated_samples = generated_samples.cpu().detach()
for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(generated_samples[i].reshape(28, 28),
cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
```

يجب أن تكون المخرجات عبارة عن أرقام تشبه بيانات التدريب، كمافي الشكل التالي:



بعد خمسين فترة من التدريب، يتم إنشاء العديد من الأرقام التي تشبه الأرقام الحقيقية. يمكنك تحسين النتائج من خلال النظرفي المزيد من فترات التدريب. كما هو الحالفي المثال السابق، باستخدام موتر عينات الفضاء الكامنة الثابتة وتغذيته للمولدفي نهاية كل فترة أثناء عملية التدريب، يمكنك تصور تطور التدريب:



يمكنك أن ترى أنهفي بداية عملية التدريب، تكون الصور التي تم إنشاؤها عشوائية تمامًا. مع تقدم التدريب، يتعلم المولد توزيع البيانات الحقيقية، وفي حوالي عشرين فترة، تشبه بعض الأرقام التي تم إنشاؤها البيانات الحقيقية بالفعل.

الاستنتاج

تهانينا! لقد تعلمت كيفية تنفيذ شبكات الخصومة التوليدية الخاصة بك. لقد مررت أولاً بمثال لعبة لفهم معمارية GAN قبل الغوص في تطبيق عملي يقوم بإنشاء صور لأرقام مكتوبة بخط اليد.

لقد رأيت أنه على الرغم من تعقيد شبكات GAN، فإن أطر التعلم الآلي مثل PyTorch تجعل التنفيذ أكثر وضوحًا من خلال تقديم الاشتقاق الآلي automatic differentiation وإعداد GPU السهل.

في هذا البرنامج التعليمي، تعلمت:

- ما هو الفرق بين النماذج التمييزية والتوليدية.
- كيف يتم تنظيم وتدريب شبكات الخصومة التوليدية.
- كيفية استخدام أدوات مثل PyTorch ووحدة معالجة الرسومات GPU لتنفيذ نماذج GAN وتدريبها.

تُعد شبكات GAN موضوعًا بحثيًا نشطًا للغاية، حيث تم اقتراح العديد من التطبيقات المثيرة في السنوات الأخيرة. إذا كنت مهتمًا بالموضوع، فتابع الأدبيات التقنية والعلمية للتحقق من أفكار التطبيقات الجديدة.

المصدر:

https://realpython.com/generative-adversarial-networks/

3) كيفية بناء شبكة الخصومة التوليدية GAN في بايثون
How to build a GAN in Python

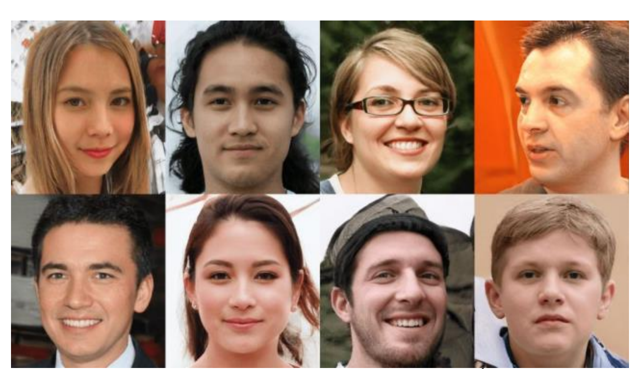
تعرف على كيفية إنشاء شبكة الخصومة التوليدية Generative Adversarial Network (GAN) عاملة بسهولةفي لغة Python، وذلك باستخدام التعلم الآلي للسماح للذكاء الاصطناعي "بإنشاء" محتوى واقعي!.

المقدمة

تعد شبكات الخصومة التوليدية (GANs) موضوعًا ساخنًافي التعلم الآلي لعدة أسباب وجيهة. فيما يلي ثلاثة من الأفضل:

1. يمكن لشبكات GAN تقديم نتائج مذهلة، وإنشاء أشياء جديدة (صور، ونصوص، وأصوات، وما إلى ذلك) عن طريق تقليد العينات التي تعرضت لها سابقًا.
2. تقدم GAN نموذجًا جديداًفي التعلم الآلي ـ نموذجًا توليديًا generative model ـ يجمع بين التقنيات الموجودة مسبقًا لتقديم أفكار ونتائج حالية وجديدة تمامًا.
3. إن شبكات GAN هي عبارة عن اكتشاف حديث (2014) لإيان جودفيلو، الباحث السابق في Google، والذي أصبح الآنفي Apple (وهو أيضًا مؤلف مشارك لمرجع قياسيفي التعلم العميق مع جوشوا بنجيو وآرون كورفيل).

من المحتمل أن يكون القراء قد واجهوا بالفعل بعض النتائج المثيرة للإعجاب التي تستطيع شبكات GAN تحقيقها، خاصةفي مجال معالجة الصور. مثل هذه الشبكات قادرة، عند الطلب، على رسم صورة لزهرة حمراء أو طائر أسود أو حتى قطة بنفسجية. علاوة على ذلك، فإن تلك الزهرة أو الطائر أو القطة غير موجودة على الإطلاقفي الواقع، ولكنها بالكامل نتاج "خيال" الشبكة.



هذه الصور ليست صورًا لأشخاص حقيقيين، فقد تم إنشاؤها بواسطة GAN مدربة بشكل مناسب!

كيف يكون هذا ممكنا، وهل يمكننا أن نشاركفي المرح؟ تحاول هذه المقالة الإجابة على كلا السؤالين، باستخدام كود Python الوظيفي الذي يمكن تشغيله على الكمبيوتر المحمول الخاص بك. قد تحتاج إلى إضافة بعض الحزم المفقودة من تثبيت Python الخاص بك، ولكن هذا هو ما يوجد من أجله Pip...

ما هي شبكة الخصومة التوليدية؟

تم تصميم الشبكات العصبية Neural networks (NNs) كنماذج للتنبؤ والتصنيف. إنها أدوات تحسين قوية وغير خطية يمكن تدريبها لتطوير معلماتها الداخلية inner parameters (أوزان الخلايا العصبية neuron weights) لتناسب بيانات التدريب. سيمكن هذا NN من التنبؤ وتصنيف البيانات غير المعروفة من نفس النوع.

نعلم جميعًا مدى الإعجاب الذي يمكن أن تكون عليه تقديرات البيانات التقريبية للشبكات العصبية، حيث يمكن أن تعني "البيانات data" أي شيء تقريبًا. ومع ذلك، فإن ميزات مثل هذه الخوارزميات تشير أيضًا إلى بعض عيوبها، مثل:

- تحتاج الشبكات العصبية إلى بيانات مسماة (مصنفة) labelled data ليتم تدريبها بشكل صحيح.
- والأسوأ من ذلك أنهم بحاجة إلى الكثير من البيانات المصنفة.
- والأسوأ من ذلك أننا عمومًا ليس لدينا أي فكرة عما تفعله محتويات الخلية العصبية فعليًا، إلافي بعض الحالات الخاصة.

في جوهرها، الشبكات العصبية هي خوارزميات خاضعة للإشراف supervised algorithms. ومع ذلك، فإن بعض متغيراتها تعمل بشكل جيد مع الخوارزميات غير الخاضعة للأشراف unsupervised algorithms. ويمكن تدريبها على أي نوع من البيانات، دون الحاجة إلى "التسمية label" الذي يتم إرفاقه عادةً لتمكين الشبكة من التمييز بين الأشياء المعروفة والأشياء غير المعروفة.

يقدم نموذج GAN إعدادًا آخر مثيرًا للاهتمام غير خاضع للإشراف للشبكات العصبية للعب فيه، ويتم وصفه بإيجاز أدناه.

دعونا نبدأ بالكلمات التي يرمز إليها اختصار GAN: التوليدية generative والعدائية adversarial والشبكات networks. الأخير هو الأكثر وضوحًا — الشبكات: يتم إنشاء شبكات GAN باستخدام شبكات عصبية (عميقة deep عادةً). تبدأ شبكة GAN بطبقة إدخال تحتوي على قدر معين من الخلايا العصبية المدخلة المتوازية (واحدة لكل رقم يمثله بيانات الإدخال)، وبعض الطبقات المخفية

وطبقة إخراج، متصلةفي رسم بياني موجه ويتم تدريبها بواسطة متغير من خوارزمية الانتشار الخلفي للتدرج الاشتقاقي gradient-descent backpropagation.

بعد ذلك نأتي إلى كلمة توليدية والتي تشير إلى هدف هذه الفئة من الخوارزميات. إنهم ينتجون البيانات بدلاً من استهلاكها. وبشكل أكثر تحديدًا، تحتوي البيانات التي تنتجها هذه الخوارزميات على معلومات جديدة من نفس "فئة class" البيانات المدخلة المستخدمةفي إنشائها. وعملية التوليد ليست عفوية، بل يتم توليد البيانات من بيانات أخرى، عبر آلية سيتم وصفها لاحقا.

وأخيرا، فإن كلمة "عدائية" ـ وهي المصطلح الأكثر غموضافي الاختصار ـ تشرح كيفية حدوث التوليد، أي من خلال المنافسة بين خصمين.في حالة GAN، فإن الخصوم هم الشبكات العصبية.

لذلك تهدف الشبكة GAN إلى توليد بيانات جديدة عبر شبكات تم إنشاؤها بشكل متعمد للتنافس مع بعضها البعض من أجل تحقيق هذا الهدف. يتم دائمًا تقسيم شبكة GAN إلى مكونين ـ شبكتين عصبيتين (عميقتين عادةً). الأول يعرف باسم المميز discriminator، ويتم تدريبه على التمييز بين مجموعة من البيانات والضوضاء النقية. على سبيل المثال، يمكن أن تتضمن البيانات المدخلة مجموعة من صور الزهور بالإضافة إلى عدد كبير من الصور الأخرى التي لا علاقة لها بالزهور. قد لا تحمل كل صورة علامة واضحة، ولكن من المعروف ما هي الصور التي تنتمي إلى مجموعة الزهور وأيها لا تنتمي.

ويمكن بعد ذلك تدريب الشبكة على التمييز بين الزهور وغير الزهور، أو،في هذا الصدد، التمييز بين الصور والصور التي تم إنشاؤها من وحدات البكسل العشوائية. هذا المكون "المميز" الأولفي شبكة GAN عبارة عن شبكة قياسية مدربة على تصنيف الأشياء. الإدخال هو مثال للبيانات التي نريد توليدها (مجموعة من صور الزهور إذا أردنا إنشاء صور زهور)، بينما الإخراج هو علامة نعم/لا.

الشبكة الأخرى هي المولد generator: وهذا ينتج كمخرجات نوع البيانات التي تم تدريب المُميِّز على تحديدها. لتحقيق هذا الإخراج، يستخدم المولد مدخلات عشوائية.في البداية، سيؤدي هذا إلى إنتاج مخرجات عشوائية، ولكن يتم تدريب المولد على نشر المعلومات بشكل عكسي، سواء كان ناتجها مشابهًا للبيانات المطلوبة أم لا (على سبيل المثال، صور الزهور).

ولتحقيق هذه الغاية، يتم تغذية تنبؤات المولد إلى المميز. يتم تدريب الأخير على التعرف على الزهور الأصلية (في هذا المثال)، لذلك إذا كان المولد قادرًا على تزييف زهرة بشكل جيد بما يكفي لخداع المميز، فيمكن لشبكة GAN الخاصة بنا إنتاج صور مزيفة للزهور التي سيلتقطها مراقب مدرب جيداً (المميز).

إحدى طرق التفكيرفي GAN هي أنها غرفة يلتقي فيها المزور forger والناقد الفني art critic: يقدم الأول لوحات مزيفة، مما يؤكد أصالتها؛ يحاول الأخير تأكيد ما إذا كانت هذه هي الصفقة الحقيقية أم

لا. إذا كان المزور ماهرًا جداًفي التزييف لدرجة أن الناقد يخلط بين اللوحات المزيفة واللوحات الأصلية، فقد يتم عرض اللوحات المزيفةفي المزاد على أمل أن يشتريها شخص ما...

للوهلة الأولى، قد تبدو شبكات GAN مشابهة للتعلم المعزز reinforcement learning، لكن التشابه الواضح لا يصمد أمام التدقيق. تقوم GAN بإنشاء شبكتين تتنافسان مع بعضهما البعض ـ الهدف هو زيادة مهاراتهما المتعارضة من أجل إنتاج بيانات مزيفة تبدو حقيقية. من ناحية أخرى، يقوم التعلم المعزز بفحص وكيل agent واحد مقابل البيئة وإما "تعزيز reinforces" أو "معاقبة punishes" الوكيل لتصحيح سلوكه. لا توجد منافسة ـ مجرد نمط يجب اكتشافه من أجل البقاء.

بدلاً من ذلك، يمكن اعتبار شبكات GAN بمثابة تعميم لمبدأ اختبار تورينج Turing test: المُميز هو المُختبر والمولد هو الآلة الراغبةفي اجتيازه، والفرق الوحيد هو أنهفي هذه الحالة كلا الممثلين عبارة عن آلات (انظر هنا لمزيد من التفاصيل حول لماذا كانت أفكار تورينج أساسية للتعلم الآلي).

GAN محلية الصنع

عادة ما تجد شبكات GAN تطبيقاتها الأكثر إثارةفي الصور المزيفة counterfeiting images، كما تمت مناقشته بالفعل. ومع ذلك، قد يتم إنشاء مقاطع فيديو ونصوص وحتى أصوات، على الرغم من أن المشكلات الفنية يمكن أن تؤدي إلى تعقيد تنفيذ "مولدات السلاسل الزمنية time series generators" هذه.

في معظم البرامج التعليمية، يتم عرض إنشاء الصور الكلاسيكية، عادةً باستخدام مجموعة بيانات MNIST لتعليم GAN كيفية كتابة الحروف والأرقام. ومع ذلك، فإن الشبكات التلافيفية convolutional networks مطلوبة لهذه العملية، وغالبًا ما يتم إهمال عنصر GAN نفسه لصالح تفاصيل حول إعداد الشبكات التلافيفية convolutional و"غير التلافيفية deconvolutional" التي تنفذ المُميِّز والمولد. بالإضافة إلى ذلك، يعد التدريب عملية طويلة جداً عند عدم وجود المعدات المناسبة (يمكن العثور على وصف لشبكات GAN هذهفي مساهمة أخرىفي مجلة Codemotion).

بدلاً من ذلك، ما يلي هو شرح لشبكة GAN بسيطة مبرمجة بلغة Python، باستخدام مكتبة Keras (والتي يمكن تشغيلها على أي كمبيوتر محمول) لتعليمها كيفية رسم فئة معينة من المنحنيات. لقد اخترت المنحني الجيبي sinusoids، ولكن أي نمط آخر من شأنه أن يعمل بشكل جيد على قدم المساواة.

أدناه، سأوضح كيفية:

- إنشاء مجموعة بيانات من sinusoids.
- إعداد شبكات المميز والمولد؛

- استخدامها لبناء شبكة GAN؛
- تدريب GAN، مع توضيح كيفية الجمع بين تدريب مكوناتها، و؛
- تأمل المنحني الجيبي sinusoid المنحرف والمشوه إلى حد ما الذي رسمه البرنامج من الضوضاء النقية.

مجموعة بيانات اصطناعية

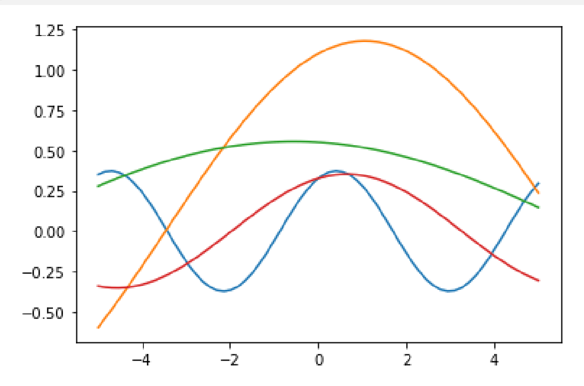
بدلًا من مجموعة من الصور، سأنتج وصفًا للمنحنيات التي أهتم بها: يمكن وصف sinusoids رياضيًا على أنها رسم بياني للدوال:

$$a \sin(bx+c)$$

حيث a، b، c هي المعلمات التي تحدد ارتفاع height المنحنى وتردده frequency ومرحلته phase. يتم رسم بعض الأمثلة على هذه المنحنياتفي الصورة التالية، والتي تم إنتاجها عبر مقتطف Python.

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.random import randint, uniform

X_MIN = -5.0
X_MAX = 5.0
X_COORDS = np.linspace(X_MIN , X_MAX, SAMPLE_LEN)
fig, axis = plt.subplots(1, 1)
for i in range(4):
    axis.plot(X_COORDS,
uniform(0.1,2.0)*np.sin(uniform(0.2,2.0)*X_COORDS +
uniform(2)))
```



نريد أن تقوم شبكة GAN الخاصة بنا بإنشاء منحنيات بهذا النوع من النماذج. لتبسيط الأمور، نعتبر a=1 وليكن $b \in [1/2,2]$ و $c \in [0,\pi]$.

أولاً، نحدد بعض الثوابت وننتج مجموعة بيانات من هذه المنحنيات. لوصف منحنى، لا نستخدم الشكل الرمزي symbolic form عن طريق دالة الجيب، بل نختار بعض النقاطفي المنحنى، مع أخذ عينات منها على نفس قيم x، ونمثل المنحنى y = f(x) بواسطة المتجه (y1،...،yN) حيث yi = f(xi) لـ xs الثابتة.

يتم إنشاء قيم y باستخدام الصيغة السابقة للقيم العشوائية لـ b وc ضمن الفواصل الزمنية المحددة. وبعد تحديد مجموعة التدريب، يمكن رسم بعض هذه المنحنيات.

```
import numpy as np
from numpy.random import uniform

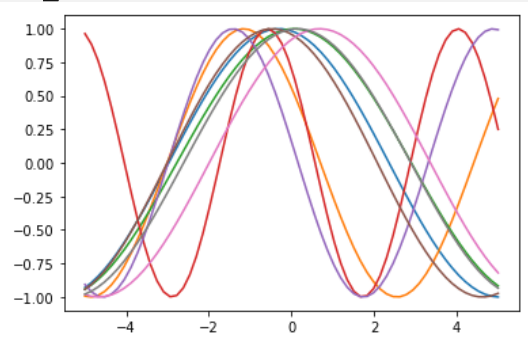
import matplotlib.pyplot as plt

SAMPLE_LEN = 64         # number N of points where a curve is
sampled
SAMPLE_SIZE = 32768     # number of curves in the training set
X_MIN = -5.0            # least ordinate where to sample
X_MAX = 5.0             # last ordinate where to sample

# The set of coordinates over which curves are sampled
X_COORDS = np.linspace(X_MIN , X_MAX, SAMPLE_LEN)

# The training set
SAMPLE = np.zeros((SAMPLE_SIZE, SAMPLE_LEN))
for i in range(0, SAMPLE_SIZE):
    b = uniform(0.5, 2.0)
    c = uniform(np.math.pi)
    SAMPLE[i] = np.array([np.sin(b*x + c) for x in
X_COORDS])

# We plot the first 8 curves
fig, axis = plt.subplots(1, 1)
for i in range(8):
    axis.plot(X_COORDS, SAMPLE[i])
```



GAN في قطع صغيرة

بعد ذلك، نحدد المميز لدينا، وهي الشبكة العصبية المستخدمة لتمييز المنحنى الجيبي عن أي مجموعة أخرى من نقاط العينة. وبالتالي يقبل المُميِّز متجه الإدخال (y1، ....، yN) ويعيد 1 إذا كان يتوافق مع منحنى جيبي، وإلا 0.

يتم بعد ذلك استخدام مكتبة Keras لإنشاء كائن Sequence يتم فيه تكديس الطبقات المختلفة للشبكة. تم ترتيب هذا المميز على شكل بيرسيبترون بسيط متعدد الطبقات multilayer perceptron، مع ثلاث طبقات: طبقة الإدخال مع N من الخلايا العصبية، N هو حجم متجهات الإدخال، وطبقة ثانية بها نفس العدد من الخلايا العصبية المخفية، وثالثة مع خلية عصبية واحدة فقط، طبقة الإخراج.

تتم تصفية مخرجات المدخلات والطبقات المخفية بواسطة دالة "relu" (التي تخفض القيم السالبة للوسيط الخاص بها x) ومن خلال "التسرب dropout" (الذي يضبط وحدات الإدخال بشكل عشوائي على 0 بتردد محدد أثناء كل خطوة من التدريب، لمنع الضبط الزائد overfitting).

يتم تنشيط الخلية العصبية الناتجة عبر دالة sigmoid التي تمتد بسلاسة من 0 إلى 1، وهما الإجابتان المحتملتان.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, LeakyReLU

DROPOUT = Dropout(0.4)          # Empirical hyperparameter
discriminator = Sequential()
discriminator.add(Dense(SAMPLE_LEN, activation="relu"))
discriminator.add(DROPOUT)
discriminator.add(Dense(SAMPLE_LEN, activation="relu"))
discriminator.add(DROPOUT)
discriminator.add(Dense(1, activation = "sigmoid"))
              discriminator.compile(optimizer = "adam", loss =
                "binary_crossentropy", metrics = ["accuracy"])
```

بعد ذلك نأتي إلى شبكة المولد. وهذا بمعنى ما مرآة للمميز؛ لا يزال لدينا ثلاث طبقات، حيث تقبل طبقة الإدخال مدخلات مشوشة noisy input بنفس حجم الإخراج (متجه مع عناصر N)، وتطبق دالة "leaky relu" (التي تخفض القيم السالبة للوسيط x إلى قيمة صغيرة مضاعف x نفسه). ومع ذلك، فإن هذه الشبكة لا تقوم بالتسرب، وتخرج النتيجة عبر دالة الظل الزائدية hyperbolic tangent. نظرًا لأن التصنيف ليس هدفنا، فإننا نستخدم متوسط مربع الخطأ mean square error كدالة الخطأ بدلاً من الإنتروبيا الثنائية cross entropy عند تدريب الشبكة واستخدامها للتنبؤات.

```
LEAKY_RELU = LeakyReLU(0.2)    # Empirical hyperparameter
generator = Sequential()
```

```
generator.add(Dense(SAMPLE_LEN))
generator.add(LEAKY_RELU)
generator.add(Dense(512))
generator.add(LEAKY_RELU)
generator.add(Dense(SAMPLE_LEN, activation = "tanh"))
  generator.compile(optimizer = "adam", loss = "mse", metrics
                                                = ["accuracy"])
```

بعد ذلك، نقوم بتوصيل مخرجات المولد إلى المميز كمدخل، بحيث تكون شبكة GAN بأكملها جاهزة للتدريب.

```
gan = Sequential()
gan.add(generator)
gan.add(discriminator)
gan.compile(optimizer = "adam", loss =
"binary_crossentropy", metrics = ["accuracy"])
```

كيف يتم تدريب GAN؟

GAN جاهز الآن للتدريب. بدلًا من إطلاق طريقة fit من Keras فورًا على كائن gan الذي أنشأناه للتو، فلنتوقف مؤقتًا ونتأمل في مفهوم GAN لفهم كيفية تدريبه بشكل صحيح.

كما ذكرنا سابقًا، يحتاج المُميِّز إلى تعلم كيفية التمييز بين المنحني الجيبي والمنحنى الآخر. يمكن القيام بذلك ببساطة عن طريق تدريبه على مجموعة بيانات SAMPLES الخاصة بنا ومجموعة بيانات مشوشة، ووضع علامات (تسميات) على العناصر في المنحنيات الجيبية sinusoids السابقة، وفي الأشكال غير المنحنيات الجيبية non-sinusoids الأخيرة.

ومع ذلك، فإن هدف المميز ليس مجرد التعرف على مجموعة البيانات الخاصة بنا، بل اعتراض المنتجات المزيفة التي ينتجها المولد. ومع أخذ ذلك في الاعتبار، يتم تدريب المُميز على النحو التالي:

1. لكل فترة epoch، يتم إجراء التدريب الدفعي batch training على كل من المميز والمولد.
2. يبدأ هذا التدريب الدفعي بمطالبة المولد بإنشاء مجموعة من المنحنيات.
3. يقترن ناتج ذلك بمجموعة من sinusoids من مجموعة بيانات SAMPLE الخاصة بنا، ويتم توفير مجموعة بيانات تحتوي على تسميات 1 (= المنحني الجيبي الحقيقي) و0 (= المنحني الجيبي التي ينتجها المولد) لتدريب المُميِّز على دفعات، وبالتالي يتم تدريبه على التعرف على المنحنيات الجيبي المتولدة من بين الأمثلة الحقيقية.
4. يتم تدريب المولد على بيانات عشوائية: ينتشر هذا التدريب بشكل عكسي على طول شبكة GAN بأكملها، ولكن يتم ترك الأوزان في المميز دون تغيير.

والنتيجة هي أن المُميِّز لم يتم تدريبه على التعرف على المنحيات الجيبية، ولكن على المميز بين المنحنيات الجيبية من مجموعات البيانات لدينا والمنحنيات الجيبية التي ينتجها المولد. وفي الوقت نفسه، يتم تدريب المولد على إنتاج المنحنيات الجيبية من البيانات العشوائية لخداع المُميِّز.

عندما يكون معدل نجاح هذا الخداع مرتفعًا (من وجهة نظر المُميِّز)، تكون شبكة GAN قادرة على توليد منحنيات جيبية. نظرًا لأننا نريد تنفيذ التعليمات البرمجية دون تجويع أجهزة الكمبيوتر المحمولة الخاصة بنا (وهو ما يمكن افتراضهفي حالة عدم وجود وحدات معالجة الرسوميات GPU وما إلى ذلك)، يتم استخدام معلمات صغيرة نسبيًا لإنتاج مجموعة البيانات الخاصة بنا وتدريب شبكة GAN. ولذلك لا يمكننا أن نتوقع أن ترسم الشبكة شكلًا جيبيًا ناعماً؛ وبدلاً من ذلك نتوقع خطًا متذبذبًا إلى حد ما والذي يعرض مع ذلك نمطًا جيبيًا.

لتوضيح كيف تبدأ GAN بالرسم بشكل عشوائي، ثم تحسن تدريجيًا مهارتهافي رسم شكل جيبي أثناء "التدريب المهني apprenticeship"، قمت برسم بعض مخرجات GAN التي تم إنشاؤها أثناء تدريبها (تم رسم 10 فترات، نظرًا لأننا نستخدم 64 فترة فقطفي المجموع).

```
EPOCHS = 64

NOISE = uniform(X_MIN, X_MAX, size = (SAMPLE_SIZE,
SAMPLE_LEN))
ONES = np.ones((SAMPLE_SIZE))
ZEROS = np.zeros((SAMPLE_SIZE))
print("epoch | dis. loss | dis. acc | gen. loss | gen. acc")
print("------+-----------+----------+-----------+----------
")

fig = plt.figure(figsize = (8, 12))
ax_index = 1
for e in range(EPOCHS):
    for k in range(SAMPLE_SIZE//BATCH):
        # Addestra il discriminatore a riconoscere le
sinusoidi vere da quelle prodotte dal generatore
        n = randint(0, SAMPLE_SIZE, size = BATCH)
        # Ora prepara un batch di training record per il
discriminatore
        p = generator.predict(NOISE[n])
        x = np.concatenate((SAMPLE[n], p))
        y = np.concatenate((ONES[n], ZEROS[n]))
        d_result = discriminator.train_on_batch(x, y)
        discriminator.trainable = False
        g_result = gan.train_on_batch(NOISE[n], ONES[n])
        discriminator.trainable = True
```

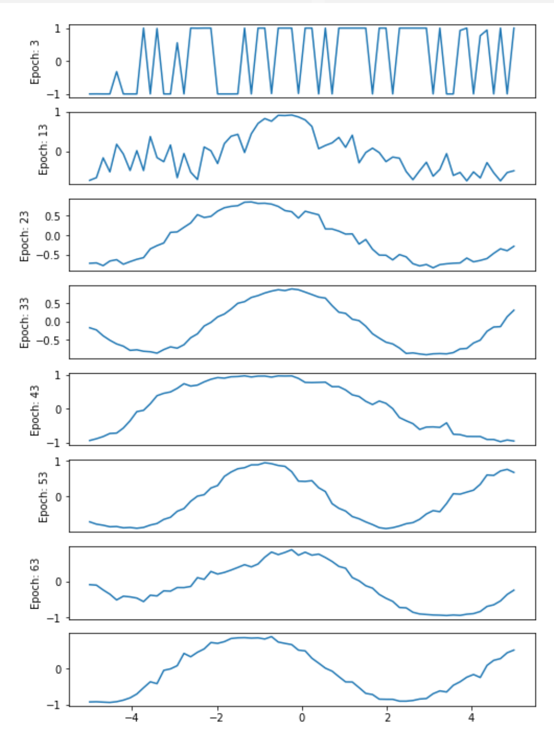
```
    print(f" {e:04n} |  {d_result[0]:.5f}  |
{d_result[1]:.5f} |  {g_result[0]:.5f}  |
{d_result[1]:.5f}")
    # At 3, 13, 23, ... plots the last generator prediction
    if e % 10 == 3:
        ax = fig.add_subplot(8, 1, ax_index)
        plt.plot(X_COORDS, p[-1])
        ax.xaxis.set_visible(False)
        plt.ylabel(f"Epoch: {e}")
        ax_index += 1

# Plots a curve generated by the GAN
y = generator.predict(uniform(X_MIN, X_MAX, size = (1,
SAMPLE_LEN)))[0]
ax = fig.add_subplot(8, 1, ax_index)
                                        plt.plot(X_COORDS, y)
```

الإخراج هو:

```
epoch | dis. loss | dis. acc | gen. loss | gen. acc
------+-----------+----------+-----------+----------
 0000 |  0.10589  |  0.96484 |  7.93257  |  0.96484
 0001 |  0.03285  |  1.00000 |  8.62279  |  1.00000
 0002 |  0.01879  |  1.00000 |  9.54678  |  1.00000
 0003 |  0.01875  |  1.00000 |  11.18307  |  1.00000
 0004 |  0.00816  |  1.00000 |  13.98673  |  1.00000
 0005 |  0.01707  |  0.99609 |  16.46034  |  0.99609
 0006 |  0.00579  |  1.00000 |  13.86913  |  1.00000
 0007 |  0.00189  |  1.00000 |  17.36512  |  1.00000
 0008 |  0.00688  |  1.00000 |  17.61729  |  1.00000
 0009 |  0.00306  |  1.00000 |  18.18118  |  1.00000
 0010 |  0.00045  |  1.00000 |  24.42766  |  1.00000
 0011 |  0.00137  |  1.00000 |  18.18817  |  1.00000
 0012 |  0.06852  |  0.98438 |  7.04744  |  0.98438
 0013 |  0.20359  |  0.91797 |  4.13820  |  0.91797
 0014 |  0.17984  |  0.93750 |  3.62651  |  0.93750
 0015 |  0.18223  |  0.91797 |  3.20522  |  0.91797
 0016 |  0.20050  |  0.91797 |  2.61011  |  0.91797
 0017 |  0.24295  |  0.90625 |  2.62364  |  0.90625
 0018 |  0.34922  |  0.83203 |  1.88428  |  0.83203
 0019 |  0.25503  |  0.88281 |  2.24889  |  0.88281
 0020 |  0.28527  |  0.88281 |  1.84421  |  0.88281
 0021 |  0.27210  |  0.88672 |  1.92973  |  0.88672
 0022 |  0.30241  |  0.88672 |  2.13511  |  0.88672
 0023 |  0.33156  |  0.82422 |  2.02396  |  0.82422
 0024 |  0.26693  |  0.86328 |  2.46276  |  0.86328
 0025 |  0.39710  |  0.82422 |  1.64815  |  0.82422
 0026 |  0.34780  |  0.83984 |  2.34444  |  0.83984
 0027 |  0.26145  |  0.90625 |  2.20919  |  0.90625
 0028 |  0.28858  |  0.86328 |  2.15237  |  0.86328
 0029 |  0.34291  |  0.83984 |  2.15610  |  0.83984
 0030 |  0.31965  |  0.86719 |  2.10919  |  0.86719
 0031 |  0.27913  |  0.89844 |  1.92525  |  0.89844
 0032 |  0.31357  |  0.87500 |  2.10098  |  0.87500
 0033 |  0.38449  |  0.83984 |  2.03964  |  0.83984
 0034 |  0.34802  |  0.81641 |  1.73214  |  0.81641
 0035 |  0.28982  |  0.87500 |  1.74905  |  0.87500
 0036 |  0.33509  |  0.85156 |  1.83760  |  0.85156
```

```
0037 |  0.29839  |  0.86719 |  1.90305  |  0.86719
0038 |  0.34962  |  0.83594 |  1.86196  |  0.83594
0039 |  0.32271  |  0.84766 |  2.21418  |  0.84766
0040 |  0.31684  |  0.84766 |  2.22909  |  0.84766
0041 |  0.37983  |  0.83984 |  1.79734  |  0.83984
0042 |  0.31909  |  0.83984 |  2.10337  |  0.83984
0043 |  0.30426  |  0.86719 |  1.98194  |  0.86719
0044 |  0.30465  |  0.86328 |  2.31558  |  0.86328
0045 |  0.35478  |  0.84766 |  2.40368  |  0.84766
0046 |  0.30423  |  0.86328 |  1.93115  |  0.86328
0047 |  0.30887  |  0.83984 |  2.17885  |  0.83984
0048 |  0.35123  |  0.86719 |  2.00351  |  0.86719
0049 |  0.24366  |  0.90234 |  2.21016  |  0.90234
0050 |  0.33797  |  0.84375 |  1.99375  |  0.84375
0051 |  0.35846  |  0.84375 |  2.17887  |  0.84375
0052 |  0.35476  |  0.83203 |  2.15312  |  0.83203
0053 |  0.28164  |  0.87109 |  2.60571  |  0.87109
0054 |  0.25782  |  0.89844 |  1.87386  |  0.89844
0055 |  0.28027  |  0.87500 |  2.30517  |  0.87500
0056 |  0.31118  |  0.84375 |  2.00939  |  0.84375
0057 |  0.32034  |  0.85547 |  2.22501  |  0.85547
0058 |  0.34665  |  0.84375 |  2.11842  |  0.84375
0059 |  0.32069  |  0.85547 |  1.79891  |  0.85547
0060 |  0.32578  |  0.87500 |  1.85051  |  0.87500
0061 |  0.32067  |  0.87109 |  1.70326  |  0.87109
0062 |  0.31929  |  0.85938 |  1.99901  |  0.85938
0063 |  0.38814  |  0.83984 |  1.55212  |  0.83984
[<matplotlib.lines.Line2D at 0x1b5c3054c48>]
```



لاحظ أن الصورة الأولى، بعد ثلاث فترات، تكون عشوائية إلى حد ما،في حين تتحرك الصور اللاحقة نحو منحنى أكثر نعومة(حتى لو كانت فتراتنا الـ 64 غير كافية لمنحنى جيد حقًا!) والأهم من ذلك، نحو منحنى أكثر نعومة. يعرض الاتجاه الجيبية.

ما يمكن ملاحظته أيضًا هو تقدم الخطأ والدقة لكل من شبكة المُميِّز والمولد بأكملها أثناء التدريب. عند فحص هذا السجل، يمكننا أن نرى أنه كلما انخفضت قيمة خطأ GAN، كان المنحنى يقترب بشكل أفضل من الشكل الجيبي. أخيرًا، عند فحص قيم المُميِّز، من الواضح أن بعض التعديلاتفي المعلمات الفائقة (أو حتىفي معمارية الشبكات) صحيحة.

الاستنتاج

المثال الذي لعبنا به هنا قد لا يبدو مثيرا للإعجاب بشكل خاص، ولكن ينبغي أن يكون كذلك حقا.في سياق هذه المقالة، تم تجميع شبكتين سطحيتين يمكن برمجتهما (بغض النظر عن dropout و leaky relu)في أواخر الثمانينات. ومع ذلك، فإن وضع هذه الشبكاتفي مواجهة بعضها البعضفي المنافسة قد أنتج شبكة توليد "ترسم" منحنيات تشبه تلك التي يتم تغذيتها بها.

علاوة على ذلك، تتعرف الشبكة على النماذج التي يجب تقليدها من مجرد وصف نموذجي صغير، ومن المحتمل أن يستغرق تشغيل البرامج على جهاز الكمبيوتر الخاص بك بضع دقائق على الأكثر.

ومن خلال الجمع بين شبكات أكثر تطورًا على نفس المنوال، يمكن إنشاء شبكة GAN قادرة على إنشاء أرقام أو أحرف أو أشكال أكثر تعقيدًا. بعض التعديلاتفي تقنيات التدريب وفي تمثيل البيانات من شأنها أن تسمح لـ GAN بإنشاء خطابات ومقاطع فيديو وفي المستقبل القريب، أي شيء يوجد الكثير من الأمثلة عليه على الويب، أي كل شيء تقريبًا!

المصدر:

https://www.codemotion.com/magazine/ai-ml/deep-learning/how-to-build-a-gan-in-python/

4) كيفية برمجة شبكة الخصومة التوليدية (GAN) في بايثون How to code a Generative Adversarial Network (GAN) in Python

الشبكات العصبية Neural networks قوية جداً.في هذه المقالة، قمنا ببرمجة شبكة عصبية من الصفر بلغة Python وإظهار كيفية استخدام الشبكات العصبية التلافيفية convolutional neural networks لتصنيف الصور. واليوم سنذهب خطوة أخرى إلى الأمام. سنتعلمفي هذا المنشور كيفية برمجة generative adversarial network (GAN) فيلغة Python لإنشاء صور مزيفة fake images. يبدو عظيما، أليس كذلك؟ لنفعلها اذا!.

تحضير السكريبت الخاص بنا على Google Colab

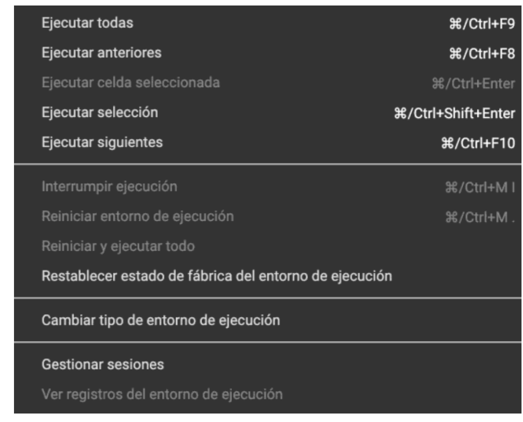
ملحوظة: إذا كنت تعرف الآن كيفية عمل Google Colab وكيف يمكنك تمكين وحدة معالجة الرسوميات GPU وحفظ/قراءة الملفات من Driveفي Colab، فتخط هذا الجزء.

كما تعلم، فإن Google Colab هي خدمة مجانية لتعلم علم البيانات. يسمح لك بشكل أساسي بتنفيذ Jupyter Notebooks المكتوبة بلغة Python على خوادم Google.

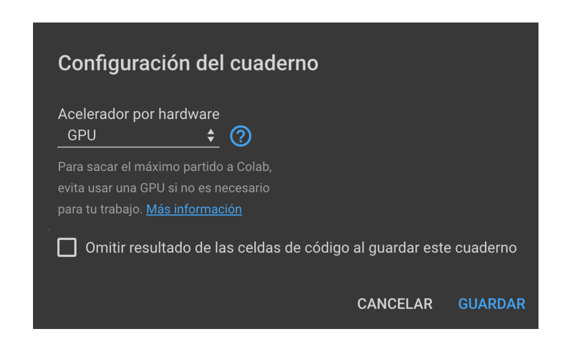
في هذا الصدد، من بين المزايا العديدة التي يقدمها Google Colab أود أن أشير إلى أنه يمكننا من تدريب شبكاتك العصبية على وحدة معالجة الرسوميات GPU مجانًا. سيؤدي ذلك إلى جعل عمليات التدريب الخاصة بك أسرع بكثير من القيام بذلك عبر وحدة المعالجة المركزية CPU، وهو أمر رائع في حالة عدم وجود جهاز كمبيوتر مزود بوحدة معالجة رسومات قوية.

لتمكين GPU على Colab، يجب عليك:

1. انتقل إلى "Change Execution Environment":



2. حدد GPU كمسرع للأجهزة hardware accelerator.



وبهذا سيكون لدينا إمكانية الوصول إلى GPU. الآن علينا أن نجعل Tensorflow يستخدمه.

وللقيام بذلك علينا تشغيل الكود التالي:

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

الآن سيتم تشغيل كل ما نصنعه على Tensorflow على وحدة معالجة الرسوميات GPU.

ومن ناحية أخرى، أوصي أيضًا بالاتصال بـ Google Drive حتى تتمكن من حفظ جميع نقاط الحفظ checkpoints والصور التي ننشئها.

وذلك لأن أحد العيوب الرئيسية لاستخدام Google Colab هو أنهفي مرحلة ما سوف يقوم بفصلك عن الخادم. لذا، إذا لم تقم بحفظ النتائج بشكل متكرر، فقد تفقد كل التقدم.

لتوصيل Colab مع Drive، تحتاج فقط إلى تشغيل سطر من التعليمات البرمجية، وإدخال عنوان URL ولصق هذا الكودفي Colab.

```
from google.colab import drive
import os
drive.mount('/content/gdrive/')
```

```
Go to this URL in a browser:
https://accounts.google.com/o/oauth2/auth?client_id=12345678
9123-
6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&
```

```
redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_ty
pe=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2faut
h%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdr
ive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos
.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeople
api.readonly

Enter your authorization code:
··········
Mounted at /content/gdrive/
```

علاوة على ذلك، تحتاج إلى اختيار مجلد Drive حيث تريد تخزين نتائجك. يمكنك القيام بذلك على النحو التالي:

```
%cd /content/gdrive/My\ Drive/Red \Neuronal \Generativa
\Antagonica
```

```
/content/gdrive/My Drive/Red Neuronal Generativa Antagonica
```

وبهذا تكون بيئتنا جاهزة بالفعل. لذا... دعونا نتعلم كيفية برمجة شبكة الخصومة التوليديةفي Python!

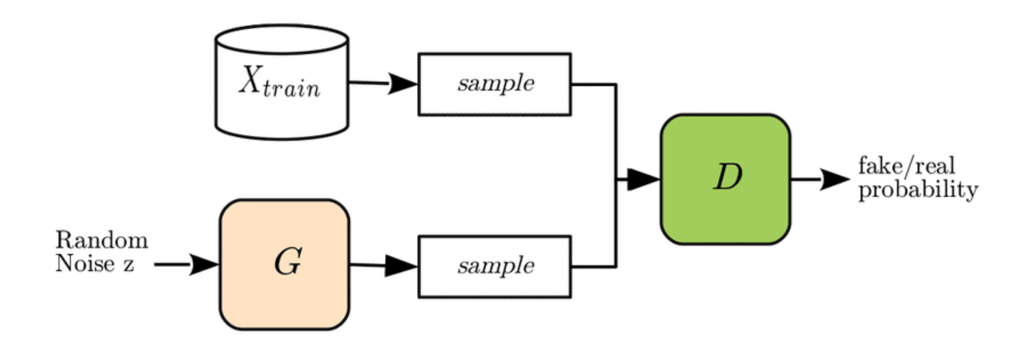
المولد/المميز هو اساس شبكة الخصومة التوليدية

لإنشاء شبكة عصبية تولد الصور، سنحتاج إلى شبكتين عصبيتين مختلفتين:

- **الشبكة التوليدية Genervative network:** تقوم هذه الشبكة العصبية بإنشاء الصور. في البداية، ستولد هذه الشبكة ضوضاء فقط، لذا سنحتاج إلى تدريبها حتى تولد صورًا أكثر واقعية.
- **الشبكة التمييزية Discriminator network:** تقوم هذه الشبكة بتصنيف ما إذا كانت الصورة حقيقية أم لا. هذه هي الشبكة التي ستمكننا من تدريب الشبكة التوليدية.

لإنشاء شبكتنا العصبية سوف نستخدم keras. ومع ذلك، نظرًا لتفرد هذه الحالة، فلن يكون الأمر سهلاً مثل إضافة الطبقات بشكل تسلسلي. بدلاً من ذلك، سيتعين علينا تنفيذ كل خطوة على حدة، حتى نتمكن لاحقًا من توصيل الشبكتين.

وفي نهاية المطاف، يتم استخدام نتائج الشبكة التمييزية من قبل الشبكة التوليدية لضبط معلماتها، كما في المثال أدناه:



ولكن، كيف يمكننا برمجة شبكة GAN هذهفي Python؟في حالتنا، سوف نستخدم مجموعة بيانات cifar10، التي تحتوي على 50000 صورة لـ 10 كائنات مختلفة بحجم 32 × 32. يمكنك التحقق من مجموعة البيانات بأكملها هنا.

في حالتنا، سنستخدم فقط صور الطائرات مجموعة البيانات، على الرغم من أن النموذج سيعمل بشكل رائع مع أي أنواع أخرى من الصور.

ومع ذلك، هل تريد أن تتعلم كيفية برمجة شبكة الخصومة التوليدية (GAN)في Python؟ دعنا نقوم به!

كيفية برمجة GAN في بايثون

كيفية برمجة الشبكة التوليدية

هيكل الشبكة التوليدية

أول شيء يتعين علينا القيام به لبرمجة كل من GAN هو معرفة معمارية كل من المولد generator والمميز discriminator. مدخلات الشبكة التوليدية هي متجه للضوضاء vector of noise. سنقوم بترقية هذه الشبكة حتى نجعلها مصفوفة بحجم 32x32x3.

فكرة الشبكة بسيطة: من خلال الضوضاء العشوائية random noise، سنقوم بدمج البيانات حتى نقوم بإنشاء صورة.في البداية، ستكون الصور ضوضاء عشوائية أيضًا، ولكن كلما قمنا بتدريب الشبكة التوليدية، كلما كانت النتائج أفضل.

الآن بعد أن أصبحت الفكرة واضحة، فلنقم ببرمجتها!

برمجة الشبكة التوليدية

أول شيء يتعين علينا القيام به هو تحميل الدوال التي سنستخدمها.في هذه الحالة سأستخدم:

- **Dense**: إنها طبقة الضوضاء الخاصة بمولدنا.
- **Conv2DTranspose**: يتيح هذا إمكانية الالتفاف للخلف، أي ترقية الصورة ودمجهافي نفس الوقت. وهو يعادل استخدام الدالة UpSampling2D متبوعة بـ Conv2D.
- **LeakyReLU**: أفضل من دوال ReLU، لأنه يتجنب اختفاء التدرج gradient vanish.
- **BatchNormalization**: يمكّن من تسوية نتيجة الالتفاف convolution. وهذا سوف يساعدنا على الحصول على نتائج أفضل. وفي حالتي لم أستخدمه لأنه بعد تجربته لم يحسن النتائج.
- **Reshape**: يتيح لنا ذلك تحويل متجه أحادي البعد إلى مصفوفة ثلاثية الأبعاد.

بالإضافة إلى ذلك علينا أن نأخذفي الاعتبار أن شكل مخرجات الشبكة التوليدية يجب أن يكون هو نفس شكل الصور الحقيقية. ولتحقيق ذلك سنستخدم الدالة Conv 2DTranspose، لكن كيف نعرف الشكل الذي سنحصل عليه؟

وفي هذا الصدد، سوف نستخدم الخطوات strides. تشير الخطوات إلى مقدار تحرك النواة من أجل الالتفاف. على سبيل المثال،في حالة صورة مقاس 18 × 18، إذا طبقنا التفافًا بنواة مكونة من 3 وخطوة مكونة من 3، فستكون النتيجة النهائية صورة مقاس 6 × 6 (18/3 x 18/3).

على العكس من ذلك، إذا أردنا زيادة حجم الصورة باستخدام Conv2DTranspose، فسنستخدم الخطوات أيضًا. على سبيل المثال، إذا أردنا الانتقال من صورة مقاس 6 × 6 إلى صورة مقاس 18 × 18، فسنحتاج إلى الحفاظ على خطوة واحدة.

من ناحية أخرى،في الطبقة الأخيرة، سنستخدم دالة tangent بحيث نحصل على قيم من ـ1 إلى 1. السبب؟ المحاولة والخطأ. لقد قمت أولاً بتجربة دالة sigmoid ولكنها لم تعمل بشكل جيد.

وأخيرًا، من المهم الإشارة إلى أننا نقوم فقط بتعريف معمارية الشبكة التوليدية. نحن لا نحاول تدريبها.

```
import keras
from keras.layers import Dense, Conv2DTranspose, LeakyReLU,
Reshape, BatchNormalization, Activation, Conv2D
from keras.models import Model, Sequential


def generador_de_imagenes():

    generador = Sequential()

    generador.add(Dense(256*4*4, input_shape = (100,)))
    #generador.add(BatchNormalization())
    generador.add(LeakyReLU())
    generador.add(Reshape((4,4,256)))
```

```
    generador.add(Conv2DTranspose(128,kernel_size=3,
strides=2, padding = "same"))
    #generador.add(BatchNormalization())
    generador.add(LeakyReLU(alpha=0.2))


    generador.add(Conv2DTranspose(128,kernel_size=3,
strides=2, padding = "same"))
    #generador.add(BatchNormalization())
    generador.add(LeakyReLU(alpha=0.2))

    generador.add(Conv2DTranspose(128,kernel_size=3,
strides=2, padding = "same"))
    #generador.add(BatchNormalization())
    generador.add(LeakyReLU(alpha=0.2))

    generador.add(Conv2D(3,kernel_size=3, padding = "same",
activation='tanh'))

    return(generador)

modelo_generador = generador_de_imagenes()

modelo_generador.summary()
```

```
Using TensorFlow backend.
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 4096)              413696
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 4096)              0
_________________________________________________________________
reshape_1 (Reshape)          (None, 4, 4, 256)         0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 8, 8, 128)         295040
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 16, 16, 128)       147584
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 32, 32, 128)       147584
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 3)         3459
=================================================================
```

```
Total params: 1,007,363
Trainable params: 1,007,363
Non-trainable params: 0
_________________________________________________________________
```

بهذا نكون قد انتهينا للتو من هيكل الشبكة التوليدية. دعونا نتحقق مما إذا كان يعمل بشكل جيد.

التحقق مما إذا كانت الشبكة التوليدية تعمل

وبما أن الشبكة لم يتم تدريبها بعد، فمن المفترض أن تقوم بإنشاء صور عشوائية. دعونا نرى ذلك.

```
import matplotlib.pyplot as plt
import numpy as np

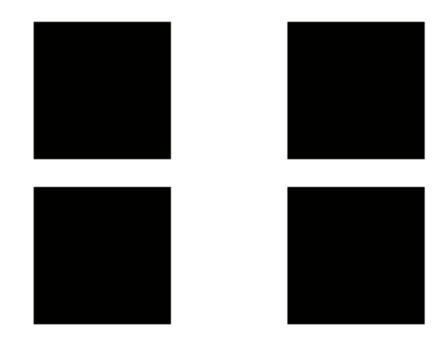
# Definir datos de entrada
def generar_datos_entrada(n_muestras):
  X = np.random.randn(100 * n_muestras)
  X = X.reshape(n_muestras, 100)
  return X

def crear_datos_fake(modelo_generador, n_muestras):
  input = generar_datos_entrada(n_muestras)
  X = modelo_generador.predict(input)
  y = np.zeros((n_muestras, 1))
  return X,y

numero_muestras = 4
X,_ = crear_datos_fake(modelo_generador, numero_muestras)

# Visualizamos resultados
for i in range(numero_muestras):
    plt.subplot(2, 2, 1 + i)
    plt.axis('off')
    plt.imshow(X[i])
```

```
Clipping input data to the valid range for imshow with RGB
data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB
data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB
data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB
data ([0..1] for floats or [0..255] for integers).
```



إنها تعمل! كما ترون، فإنه يولد الضوضاء فقط، ولكن هذا بالضبط ما توقعناه. على أية حال، لقد انتهينا للتو من الخطوة الأولى لإنشاء شبكة GAN باستخدام Python. الآن دعونا نرى كيفية إنشاء الشبكة التمييزية!

برمجة الشبكة التمييزية

هيكل الشبكة التمييزية

شبكتنا التمييزية هي شبكة عصبية تلافيفية convolutional neural network عادية. سوف يستغرق صورة كمدخل وسيقوم الإخراج بإرجاع قيمة ثنائية.

على الرغم من وجود شبكة تلافيفية، فمن المستحسن استخدام الطبقات المتسربة dropout layers بعد كل تلافيف لتجنب الضبط الزائد overfitting، وفي حالتي فقد استخدمتها فقط على الطبقة الأخيرة. مرة أخرى فعلت ذلك بعد أن حاولت ورأيت أنه لم يحسن أداء النموذج.

من ناحية أخرى، على هذه الشبكة، سوف نستخدم دالة التنشيط sigmoid في الطبقة الأخيرة. كما علقت على هذا المنشور، عندما نريد تصنيف صورتين، فإن دوال التنشيط sigmoid تعطي احتمالية أن تكون الصورة من المجموعة المستهدفة.

عندما يتعلق الأمر بالمُحسِّن، سنستخدم مُحسِّن Adam، لأنه يعمل بشكل جيد بشكل خاص مع مجموعات البيانات الكبيرة وعادةً ما يعمل بشكل أفضل من التدرج الاشتقاقي Gradient Descent.

علاوة على ذلك، على الرغم من أننا عادةً ما نستخدم معدل التعلم learning rate الافتراضي والإصدارات التجريبية، في هذه الحالة، سنتبع توصيات هذه الورقة، والتي تشير إلى تعيين معدل تعلم قدره 0.0002 وبيتا قدره 0.5.

```
from keras.layers import Conv2D, Flatten, Dropout
from keras.optimizers import Adam

def discriminador_de_imagenes():
```

```
    discriminador = Sequential()
    discriminador.add(Conv2D(64, kernel_size=3, padding =
"same", input_shape = (32,32,3)))
    discriminador.add(LeakyReLU(alpha=0.2))
    #discriminador.add(Dropout(0.2))

    discriminador.add(Conv2D(128,
kernel_size=3,strides=(2,2), padding = "same"))
    discriminador.add(LeakyReLU(alpha=0.2))
    #discriminador.add(Dropout(0.2))

    discriminador.add(Conv2D(128,
kernel_size=3,strides=(2,2), padding = "same"))
    discriminador.add(LeakyReLU(alpha=0.2))
    #discriminador.add(Dropout(0.2))

    discriminador.add(Conv2D(256, kernel_size=3,
strides=(2,2), padding = "same"))
    discriminador.add(LeakyReLU(alpha=0.2))
    #discriminador.add(Dropout(0.2))

    discriminador.add(Flatten())
    discriminador.add(Dropout(0.4))
    discriminador.add(Dense(1, activation='sigmoid'))

    opt = Adam(lr=0.0002 ,beta_1=0.5)
    discriminador.compile(loss='binary_crossentropy',
optimizer= opt , metrics = ['accuracy'])

    return(discriminador)

modelo_discriminador = discriminador_de_imagenes()
modelo_discriminador.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 64)        1792
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 256)         295168
_________________________________________________________________
```

```
leaky_re_lu_8 (LeakyReLU)     (None, 4, 4, 256)         0
_________________________________________________________________
flatten_1 (Flatten)           (None, 4096)              0
_________________________________________________________________
dropout_1 (Dropout)           (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)               (None, 1)                 4097
=================================================================
Total params: 522,497
Trainable params: 522,497
Non-trainable params: 0
_________________________________________________________________
```

بهذا نكون قد انتهينا للتو من هيكل المُميِّز. ومع ذلك، لا تزال هناك بعض الأمور المهمة التي ينبغي علينا القيام بها:

- إنشاء شبكة GAN الخاصة بنا: لقد قمنا حتى الآن بإنشاء الشبكتين، لكنهما غير متصلتين بعد. سنحتاج إلى ربطهما معًا.
- تحديد دالة التدريب: هذا سيجعل شبكتنا التوليدية تتعلم.في هذه الخطوة سنقوم أيضًا بتضمين النقطة التالية.
- حفظ نتائج شبكتنا: سنقوم بحفظ أوزان الشبكة والصور التي تولدها.في النهاية، نريد جميعًا أن نتحمّس لرؤية كيف تتدرب الشبكة وتتحسن.

كما ترون، لا يزال هناك الكثير من الأشياء التي يتعين القيام بها، لذلك دعونا ننجزها!

الخطوات الأخيرة لإنشاء GAN في بايثون

تحميل البيانات من Cifar10

لتدريب شبكة GAN الخاصة بنا، نحتاج أولاً إلى تحميل مجموعة البيانات من Cifar10.

علاوة على ذلك، سنقوم بتسوية البيانات normalize the data. وهذا سيجعل النموذج يعمل بشكل أسرع. للقيام بذلك، عندما تنتقل طبقة RGB من 0 إلى 255، سنطرح ثم نقسم 127.5. وبذلك، ستنتقل القيم من ــ1 إلى 1.

```
from keras.datasets import cifar10

def cargar_imagenes():
    (Xtrain, Ytrain), (_, _) = cifar10.load_data()

    # Nos quedamos con los perros
    indice = np.where(Ytrain == 0)
    indice = indice[0]
    Xtrain = Xtrain[indice, :,:,:]
```

```
    # Normalizamos los datos
    X = Xtrain.astype('float32')
    X = (X - 127.5) / 127.5

    return X

print(cargar_imagenes().shape)
```

```
Downloading data from
https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 6s
0us/step
(5000, 32, 32, 3)
```

تدريب الشبكة التمييزية

الآن بعد أن قمنا بتحميل صورنا، علينا تدريب الشبكة التمييزية لدينا. للقيام بذلك، سنحتاج إلى صور حقيقية real ومزيفة fake، لذلك سنقوم بإنشاء دالة تولد ذلك بالضبط.

من أجل إنشاء صور مزيفة، من المهم إنشاء صور لها نفس شكل الصور الحقيقية. كما رأينا من قبل، الصور الحقيقية هي مصفوفات بأبعاد 32x32x3، لذلك سنحتاج إلى إنشاء عدد n من الصور بهذا الشكل.

علاوة على ذلك، ينصح البعض بإعطاء العلامة 0 للصور المزيفة لتحسين الأداء. بعد عدة محاولات، رأيت تحسنًا طفيفًا من خلال القيام بذلك، لذلك قمنا بتعيين الصور المزيفة كتسمية 0.

```
import random

def cargar_datos_reales(dataset, n_muestras):
  ix = np.random.randint(0, dataset.shape[0], n_muestras)
  X = dataset[ix]
  y = np.ones((n_muestras, 1))
  return X,y

def cargar_datos_fake(n_muestras):
  X = np.random.rand(32 * 32 * 3 * n_muestras)
  X = -1 + X * 2
  X = X.reshape((n_muestras, 32,32,3))
  y = np.zeros((n_muestras, 1))
  return X,y
```

الآن وقد أصبح لدينا مولد الصور الحقيقية والمزيفة وسنستخدمه لتدريب الشبكة التمييزية الخاصة بنا.

من المهم تدريب المُميِّز مسبقًا لأنه عندما نقوم بتدريب GAN سنقوم فقط بتدريب المولد، وليس المُميِّز.

للتأكد من أن المميز قد تدربت بشكل صحيح، سنحصل على دقة البيانات الحقيقية والمزيفة ونرى ما إذا كانت تتحسن أم لا.

للتدريب، سنمرر نصف البيانات المزيفة والنصف الآخر من البيانات الحقيقية. ومن ثم نحسب medio_batch:

```
def entrenar_discriminador(modelo, dataset,
n_iteraciones=20, batch = 128):
  medio_batch = int(batch/2)

  for i in range(n_iteraciones):
    X_real, y_real = cargar_datos_reales(dataset,
medio_batch)
    _, acc_real = modelo.train_on_batch(X_real, y_real)

    X_fake, y_fake = cargar_datos_fake(medio_batch)
    _, acc_fake = modelo.train_on_batch(X_fake, y_fake)

    print(str(i+1) + ' Real:' + str(acc_real*100) + ',
Fake:' + str(acc_fake*100))
```

الآن بعد أن حصلنا على كل قطعنا، علينا أن نجمعها معًا لتدريب الشبكة التمييزية لدينا:

```
dataset = cargar_imagenes()
entrenar_discriminador(modelo_discriminador, dataset)
```

```
1 Real:78.125, Fake:1.5625
2 Real:96.875, Fake:1.5625
3 Real:95.3125, Fake:26.5625
4 Real:98.4375, Fake:56.25
5 Real:95.3125, Fake:96.875
6 Real:85.9375, Fake:100.0
7 Real:92.1875, Fake:100.0
8 Real:75.0, Fake:100.0
9 Real:78.125, Fake:100.0
10 Real:84.375, Fake:100.0
11 Real:93.75, Fake:100.0
12 Real:92.1875, Fake:100.0
13 Real:98.4375, Fake:100.0
14 Real:100.0, Fake:100.0
15 Real:98.4375, Fake:100.0
16 Real:95.3125, Fake:100.0
17 Real:100.0, Fake:100.0
18 Real:100.0, Fake:100.0
19 Real:100.0, Fake:100.0
20 Real:98.4375, Fake:100.0
```

لقد قمنا للتو بتدريب شبكتنا التمييزية! كان ذلك سهلاً، أليس كذلك؟ الآن بعد أن أنشأنا كلاً من الشبكة التوليدية والشبكة التمييزية، فلنقم ببرمجة شبكة الخصومة التوليدية (GAN) الخاصة بنا في Python!

برمجة شبكة الخصومة التوليدية لدينا

الآن بعد أن أصبح لدينا جميع أجزاء شبكة الخصومة التوليدية، علينا أن نجمعها معًا بحيث يقوم المولد بإنشاء صور ويقوم المميز بتصنيف ما إذا كانت الصور حقيقية أم لا.

وبما أن المميز قد تم تدريبها بالفعل، فسوف نقوم بتعيين المعلمة القابلة للتدريب على أنها FALSE. علاوة على ذلك، ستكون دالة التكلفة (الخطأ) binary_crossentropy، حيث ستساعدنا في تصنيف الصور بين مزيفة (0) وحقيقية (1).

```
def crear_gan(discriminador, generador):
    discriminador.trainable=False
    gan = Sequential()
    gan.add(generador)
    gan.add(discriminador)

    opt = Adam(lr=0.0002,beta_1=0.5)
    gan.compile(loss = "binary_crossentropy", optimizer =
opt)

    return gan

gan = crear_gan(modelo_discriminador,modelo_generador)
gan.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_1 (Sequential)    (None, 32, 32, 3)         1007363
_________________________________________________________________
sequential_2 (Sequential)    (None, 1)                 522497
=================================================================
Total params: 1,529,860
Trainable params: 1,007,363
Non-trainable params: 522,497
_________________________________________________________________
```

لقد قمنا للتو ببرمجة شبكة الخصومة التوليدية (GAN) في Python! الآن علينا فقط تحديد حلقة التدريب الخاصة بالشبكة وبعض الأشياء الأخرى التي نريد التحقق منها أثناء عملية التدريب، مثل الدقة أو الصور التي تم إنشاؤها.

للقيام بذلك، سأقوم بإنشاء دالتين: واحدة لحساب وحفظ الخطأ وأوزان النموذج والأخرى لحفظ الصور التي تم إنشاؤها.

دوال تقييم النماذج وتوليد الصور

سنبدأ بالدالة التي تحفظ نتائج النموذج. للقيام بذلك، ونظرًا لمجموعة من الصور التي تم إنشاؤها، سأقوم بتخزين 10 منها. وهكذا، تمكنت من رؤية كيفية تحسين GAN وتمكنت أيضًا من إنشاء ملف GIFفي هذا المنشور.

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from datetime import datetime

def mostrar_imagenes_generadas(datos_fake, epoch):

  now = datetime.now()
  now = now.strftime("%Y%m%d_%H%M%S")

  # Hacemos que los datos vayan de 0 a 1
  datos_fake = (datos_fake + 1) / 2.0

  for i in range(10):
    plt.imshow(datos_fake[i])
    plt.axis('off')
    nombre = str(epoch) + '_imagen_generada_' + str(i) +
'.png'
    plt.savefig(nombre, bbox_inches='tight')
    plt.close()
```

علاوة على ذلك، من المهم أيضًا حفظ النموذج أثناء تدريبه. يمكن لـ Google Colab إنهاء الجلسات، لذا، إذا لم نحفظ نقاط الحفظ checkpoints، فقد نفقد عملية التدريب بأكملها.

ومن ناحية أخرى، سوف نقوم بتقييم أداء شبكة التصنيف. وللقيام بذلك، سنقوم بإنشاء بيانات جديدة يمكننا من خلالها تقييم النموذج. لماذا لا تستخدم البيانات التي أنشأناها بالفعلفي تلك الدفعة التدريبية؟ حسنًا... لأنه من الأفضل التدريب على البيانات الجديدة بدلاً من القيام بذلك باستخدام البيانات التي تدرب عليها النموذج للتو.

أخيرًا، نظرًا لأن هذه الدالة تولد صورًا جديدة، فسنضيف دالة التصور داخل هذه الدالة.

```
def evaluar_y_guardar(modelo_generador, epoch,
medio_dataset):

  # We save the model
  now = datetime.now()
  now = now.strftime("%Y%m%d_%H%M%S")
  nombre = str(epoch) + '_' + str(now)+"_modelo_generador_"
+ '.h5'
  modelo_generador.save(nombre)
```

```
  # We generate new data
  X_real,Y_real = cargar_datos_reales(dataset,
medio_dataset)
  X_fake, Y_fake =
crear_datos_fake(modelo_generador,medio_dataset)

  # We evaluate the model
  _, acc_real = modelo_discriminador.evaluate(X_real,
Y_real)
  _, acc_fake = modelo_discriminador.evaluate(X_fake,
Y_fake)

  print('Acc Real:' + str(acc_real*100) + '% Acc Fake:' +
str(acc_fake*100)+'%')
```

الآن بعد أن أصبح لدينا جميع الدوال التي نحتاجها، يمكننا إنشاء دالة التدريب:

```
def entrenamiento(datos, modelo_generador,
modelo_discriminador, epochs, n_batch, inicio = 0):
  dimension_batch = int(datos.shape[0]/n_batch)
  medio_dataset = int(n_batch/2)

  # We iterate over the epochs
  for epoch in range(inicio, inicio + epochs):
    # We iterate over all batches
    for batch in range(n_batch):

      # We load all the real data
      X_real,Y_real = cargar_datos_reales(dataset,
medio_dataset)


      # We train the discriminator withEnrenamos
discriminador con datos reales
      coste_discriminador_real, _ =
modelo_discriminador.train_on_batch(X_real, Y_real)
      X_fake, Y_fake =
crear_datos_fake(modelo_generador,medio_dataset)

      coste_discriminador_fake, _ =
modelo_discriminador.train_on_batch(X_fake, Y_fake)

      # We generate input images for the GAN
      X_gan = generar_datos_entrada(medio_dataset)
      Y_gan = np.ones((medio_dataset, 1))

      # We train the GAN with fake data
      coste_gan = gan.train_on_batch(X_gan, Y_gan)
```

```
    # Every 10 epochs we show the results and cost
    if (epoch+1) % 10 == 0:
      evaluar_y_guardar(modelo_generador,epoch = epoch,
medio_dataset= medio_dataset)
      mostrar_imagenes_generadas(X_fake, epoch = epoch)
```

مع هذا، دعونا نرى ما إذا كانت شبكة GAN تعمل!

```
entrenamiento(dataset, modelo_generador,
modelo_discriminador, epochs = 300, n_batch=128, inicio = 0)
```

```
/usr/local/lib/python3.6/dist-
packages/keras/engine/training.py:297: UserWarning:
Discrepancy between trainable weights and collected
trainable weights, did you set `model.trainable` without
calling `model.compile` after ?
  'Discrepancy between trainable weights and collected
trainable'
```

```
64/64 [==============================] - 0s 2ms/step
64/64 [==============================] - 0s 225us/step
Acc Real:79.6875% Acc Fake:84.375%
64/64 [==============================] - 0s 224us/step
64/64 [==============================] - 0s 199us/step
Acc Real:65.625% Acc Fake:95.3125%
64/64 [==============================] - 0s 247us/step
64/64 [==============================] - 0s 207us/step
Acc Real:67.1875% Acc Fake:87.5%
64/64 [==============================] - 0s 220us/step
64/64 [==============================] - 0s 200us/step
Acc Real:76.5625% Acc Fake:78.125%
64/64 [==============================] - 0s 239us/step
64/64 [==============================] - 0s 187us/step
Acc Real:75.0% Acc Fake:78.125%
64/64 [==============================] - 0s 230us/step
64/64 [==============================] - 0s 195us/step
Acc Real:70.3125% Acc Fake:78.125%
64/64 [==============================] - 0s 208us/step
64/64 [==============================] - 0s 191us/step
Acc Real:73.4375% Acc Fake:90.625%
64/64 [==============================] - 0s 216us/step
64/64 [==============================] - 0s 193us/step
Acc Real:71.875% Acc Fake:87.5%
64/64 [==============================] - 0s 247us/step
64/64 [==============================] - 0s 186us/step
Acc Real:76.5625% Acc Fake:89.0625%
64/64 [==============================] - 0s 236us/step
64/64 [==============================] - 0s 195us/step
Acc Real:78.125% Acc Fake:90.625%
64/64 [==============================] - 0s 256us/step
64/64 [==============================] - 0s 202us/step
Acc Real:84.375% Acc Fake:89.0625%
64/64 [==============================] - 0s 218us/step
64/64 [==============================] - 0s 199us/step
Acc Real:85.9375% Acc Fake:96.875%
```

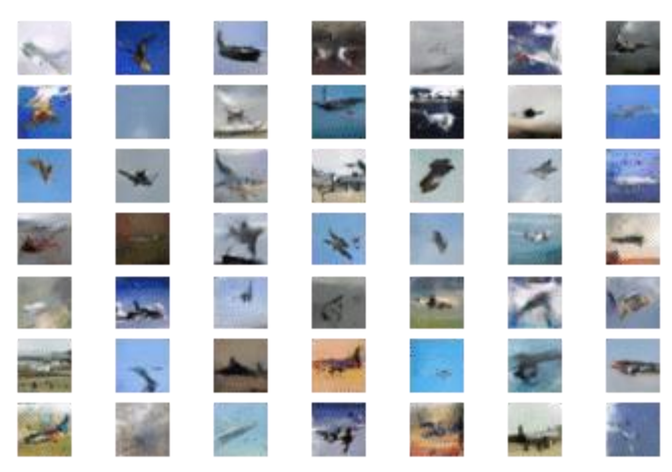
```
64/64 [==============================] - 0s 233us/step
64/64 [==============================] - 0s 198us/step
Acc Real:90.625% Acc Fake:96.875%
64/64 [==============================] - 0s 258us/step
64/64 [==============================] - 0s 206us/step
Acc Real:90.625% Acc Fake:93.75%
64/64 [==============================] - 0s 277us/step
64/64 [==============================] - 0s 230us/step
Acc Real:95.3125% Acc Fake:98.4375%
64/64 [==============================] - 0s 251us/step
64/64 [==============================] - 0s 194us/step
Acc Real:98.4375% Acc Fake:96.875%
64/64 [==============================] - 0s 236us/step
64/64 [==============================] - 0s 202us/step
Acc Real:98.4375% Acc Fake:96.875%
64/64 [==============================] - 0s 225us/step
64/64 [==============================] - 0s 197us/step
Acc Real:96.875% Acc Fake:93.75%
64/64 [==============================] - 0s 246us/step
64/64 [==============================] - 0s 203us/step
Acc Real:96.875% Acc Fake:98.4375%
64/64 [==============================] - 0s 238us/step
64/64 [==============================] - 0s 233us/step
Acc Real:98.4375% Acc Fake:100.0%
64/64 [==============================] - 0s 262us/step
64/64 [==============================] - 0s 208us/step
Acc Real:95.3125% Acc Fake:100.0%
64/64 [==============================] - 0s 310us/step
64/64 [==============================] - 0s 259us/step
Acc Real:98.4375% Acc Fake:95.3125%
64/64 [==============================] - 0s 275us/step
64/64 [==============================] - 0s 205us/step
Acc Real:98.4375% Acc Fake:98.4375%
64/64 [==============================] - 0s 268us/step
64/64 [==============================] - 0s 228us/step
Acc Real:96.875% Acc Fake:98.4375%
64/64 [==============================] - 0s 227us/step
64/64 [==============================] - 0s 191us/step
Acc Real:100.0% Acc Fake:96.875%
64/64 [==============================] - 0s 276us/step
64/64 [==============================] - 0s 212us/step
Acc Real:100.0% Acc Fake:98.4375%
64/64 [==============================] - 0s 220us/step
64/64 [==============================] - 0s 200us/step
Acc Real:100.0% Acc Fake:100.0%
64/64 [==============================] - 0s 260us/step
64/64 [==============================] - 0s 217us/step
Acc Real:100.0% Acc Fake:95.3125%
64/64 [==============================] - 0s 207us/step
64/64 [==============================] - 0s 207us/step
Acc Real:100.0% Acc Fake:98.4375%
64/64 [==============================] - 0s 258us/step
64/64 [==============================] - 0s 207us/step
Acc Real:96.875% Acc Fake:98.4375%
```

لقد تعلمنا للتو برمجة GANفي Python وقمنا بتدريبها للتو! دعونا نرى كيف هي الصور التي تم إنشاؤها!

```
X_fake, _ = crear_datos_fake(n_muestras=49,
modelo_generador=modelo_generador)
```

```
X_fake = (X_fake+1)/2

for i in range(49):
  plt.subplot(7,7,i+1)
  plt.axis('off')
  plt.imshow(X_fake[i])
```



إنها تعمل! على الرغم من الدقة المنخفضة (في النهاية، فهي ذات دقة منخفضة جداً)، قامت شبكتنا العصبية بتوليد صور تشبه الطائرات! نعم، صحيح أنه ليست كل الصور تبدو وكأنها طائرات. لو قمنا بتدريب الشبكة لمزيد من الفترات لكانت النتائج أفضل.

على أية حال، لقد تعلمت للتو برمجة شبكة GAN بلغة Python والتي تولد صورًا مزيفة ولكن واقعية!

إذا كنت ترغب في تدريب هذا النوع من الشبكات باستخدام بيانات أخرى، فدعني أقدم لك بعض النصائح.

4 نصائح لبرمجة شبكة الخصومة التوليدية (GAN) في بايثون

1. قم بإنشاء نوع واحد من الصور

في البداية حاولت إنشاء شبكة تولد صورًا مثل تلك الموجودة في مجموعة بيانات Cifar 10. ومع ذلك، فإن هذا يعني أن الشبكة يجب أن تكون قادرة على إنشاء 10 أنواع مختلفة من الصور... وهو أمر معقد للغاية.

لذا، إذا كنت ترغب في تدريب GAN، فإنني أنصحك بالقيام بذلك لشيء واحد فقط. كلاب، طائرات، وجوه... كل ما تريد، ولكن نوع واحد فقط من الصور.

2. افشل بسرعة وتحسن

في البداية، تركت النماذج تتدرب لمدة تتراوح بين 100 إلى 200 فترة قبل التحقق من كيفية عملها. مما لا شك فيه أن هذا جعل عمليات التعلم أبطأ بكثير، لأن تعلم النموذج يستغرق من 10 إلى 20 دقيقة.

لذا، إذا كنت تريد إجراء بعض الاختبارات، فإنني أوصي بتدريب النموذج لمدة 20 إلى 30 فترة فقط. إذا رأيت أن التغيير الذي قمت بتنفيذه لا يتحسن... أوقف عملية التعلم وجرب شيئًا آخر.

3. حدد المقياس لتقييم النموذج الخاص بك

عندما ندرب شبكة عصبية عادية، يكون من الواضح تمامًا ما هو المؤشر الذي نحتاج إلى تصوره. ولكن ماذا عن GAN؟

في هذا المنشور، وجدت أفضل المؤشرات التي يجب التحقق منها عند تدريب شبكة GAN. وفي حالتي استخدمت الدقة مع التركيز على دقة الصور المزيفة.

4. إذا انتهت الجلسة... قم بتحميل النموذج الخاص بك

كما أوضحت من قبل، أحد المفاتيح هو حفظ نموذج المولد. ومن خلال القيام بذلك، إذا انتهت الجلسة، يمكنك دائمًا تحميل آخر نموذج تم تدريبه واستخدامه لتجنب الاضطرار إلى إعادة تدريبه من البداية.

لتحميل النموذج بسيط جدا. يجب عليك:

1) انتقل إلى مسار Drive الصحيح.
2) قم بتحميل النموذج باستخدام الدالة Load_model.

أعرض لك ومثالا:

```
# 1. We go to the correct folder in Drive
from google.colab import drive
import os
drive.mount('/content/gdrive/')
%cd /content/gdrive/My\ Drive/Red \Neuronal \Generativa
\Antagonica

# 2. We import the model
from keras.models import load_model

modelo_generador =
load_model('299_20200712_104051_modelo_generador_.h5')
```

```
/usr/local/lib/python3.6/dist-
packages/keras/engine/saving.py:341: UserWarning: No
training configuration found in save file: the model was
*not* compiled. Compile it manually.
  warnings.warn('No training configuration found in save
file:
```

المصدر:

https://anderfernandez.com/en/blog/how-to-code-gan-in-python/

5) توليد الارقام المكتوبة بخط اليد MNIST باستخدام شبكات الخصومة التوليدية MNIST Handwritten Digits Generation using GANs

GAN (شبكة الخصومة التوليدية Generative Adversarial Network) هو إطار عمل اقترحه إيان جودفيلو ويوشوا بينجيو وآخرونفي عام 2014.

يمكن تدريب GAN على إنشاء صور من الضوضاء العشوائية random noises. على سبيل المثال، يمكننا تدريب GAN على MNIST (مجموعة بيانات الأرقام المكتوبة بخط اليد hand-written digits dataset) لإنشاء صور رقمية تبدو وكأنها صور رقمية مكتوبة بخط اليد من MNIST، والتي يمكن استخدامها لتدريب الشبكات العصبية الأخرى.

يعتمد الكود الموجودفي هذا notebook على مثال GAN MNIST في TensorFlow بواسطة Udacity والذي يستخدم TensorFlow، ولكننا نستخدم Keras أعلى TensorFlow لإنشاء شبكات أكثر وضوحًا.

MNIST

MNIST هي قاعدة بيانات معروفة للأرقام المكتوبة بخط اليد.

```
import numpy as np
import keras
import keras.backend as K
from keras.layers import Input, Dense, Activation,
LeakyReLU, BatchNormalization
from keras.models import Sequential
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
```

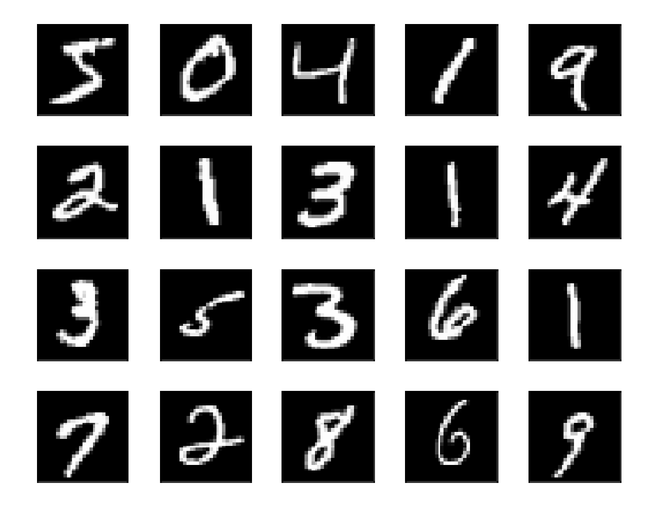
```
Using TensorFlow backend.
```

أدناه سيتم تنزيل مجموعة بيانات MNIST.

```
(X_train, y_train), (X_test, y_test) =
keras.datasets.mnist.load_data()
```

دعونا نفحص عينة من الصور. نحن نستخدم خريطة الألوان "gray" لأنها لا تحتوي على معلومات الألوان.

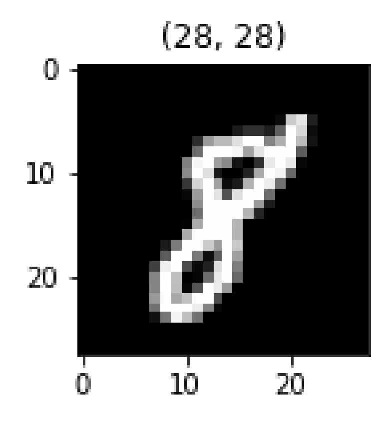
```
plt.figure(figsize=(5, 4))
for i in range(20):
    plt.subplot(4, 5, i+1)
    plt.imshow(X_train[i], cmap='gray')
    plt.xticks([])
    plt.yticks([])
plt.tight_layout()
plt.show()
```



جميع الصور الرقمية من MNIST تأتي بحجم 28 × 28.

```
sample = X_train[17]

plt.figure(figsize=(3, 2))
plt.title(sample.shape)
plt.imshow(sample, cmap='gray')
plt.show()
```



الحد الأدنى والحد الأقصى لحجم بيانات صورة MNIST هو 0 و255 على التوالي.

```
X_train.min(), X_train.max()
```

```
(0, 255)
```

المولد

نريد إنشاء مولد يقوم بإنشاء صور واقعية مكتوبة بخط اليد.

يُطلق على المدخلات إلى المولد اسم "العينة الكامنة latent sample" وهي عبارة عن سلسلة من الأرقام التي تم إنشاؤها عشوائيًا. نحن نستخدم التوزيع الطبيعي بدلا من التوزيع المنتظم.

```
def make_latent_samples(n_samples, sample_size):
    #return np.random.uniform(-1, 1, size=(n_samples,
sample_size))
    return np.random.normal(loc=0, scale=1, size=(n_samples,
sample_size))
```

حجم العينة sample siz هو معلمة فائقة hyperparameter. أدناه، نستخدم متجهًا مكونًا من 100 رقم تم إنشاؤه عشوائيًا كعينة.

```
make_latent_samples(1, 100) # generates one sample
```

```
array([[ 0.27473292, -0.10655303,  1.09659154, -0.15366093, -0.61587259,
         0.26621725, -0.01077858,  1.90989847, -1.18569009, -0.03249574,
         0.6695888 ,  0.93482157, -1.00884217, -0.29230866, -1.13929507,
        -0.01707456,  0.37335705,  0.33970841,  1.37859658, -0.53026397,
         1.12131043, -1.66056623, -0.01493777,  0.808652  , -0.36080177,
         1.18045669, -0.33343053,  0.60287437,  0.9825658 , -0.31889656,
         0.1179318 , -0.91808507, -1.03957979,  0.40643158, -0.32754068,
         0.80441626, -0.01966988, -0.79409032,  1.55127879,  0.34457065,
        -0.00255198,  1.01100422,  0.56261678, -0.39284342, -0.03217389,
         1.09418122, -0.90881511, -0.19342759,  0.3317994 ,  0.19549762,
        -1.40058816, -0.16028498, -1.86537691,  0.6165322 , -0.4672151 ,
        -0.23835781, -0.35751269, -0.97823372,  1.26912872, -1.29290883,
        -0.97779726,  1.76487061, -0.33914689, -0.57437618,  0.86655979,
        -0.27751868,  0.71869572, -1.22436248,  0.7134086 , -1.1244994 ,
         0.99746905, -0.00786507, -0.66620361, -1.40849483, -0.26476278,
        -0.40399178,  0.35693832, -1.45288997, -0.79326825, -0.48003003,
        -0.15437594,  0.12191884, -0.00680743, -1.30782153,  0.45268918,
         0.68991131,  1.85151145,  1.05853026, -0.4318387 , -0.19847975,
        -1.13489859, -0.59428163,  0.4483107 , -1.07967249, -0.02395855,
         1.00938931,  0.06230197, -1.74878905, -0.13811637, -0.4902213 ]])
```

المولد generator عبارة عن شبكة عصبية بسيطة متصلة بالكامل fully connected neural network بطبقة مخفية hidden layer واحدة مع تنشيط leaky ReLU. يأخذ عينة كامنة واحدة (100 قيمة) وينتج 784 (= 28 × 28) نقطة بيانات تمثل صورة رقمية.

```
generator = Sequential([
```

```
    Dense(128, input_shape=(100,)),
    LeakyReLU(alpha=0.01),
    Dense(784),
    Activation('tanh')
], name='generator')

generator.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               12928
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 784)               101136
_________________________________________________________________
activation_1 (Activation)    (None, 784)               0
=================================================================
Total params: 114,064
Trainable params: 114,064
Non-trainable params: 0
_________________________________________________________________
```

التنشيط الأخير هو tanh. وهذا يعني أيضًا أننا بحاجة إلى إعادة قياس صور MNIST لتكون بين ـ1 و1.

في البداية، يمكن للمولد إنتاج القمامة فقط.

على هذا النحو، يحتاج المولد إلى تعلم كيفية إنشاء صور واقعية مكتوبة بخط اليد من العينة الكامنة (الأرقام التي يتم إنشاؤها عشوائيًا).

كيفية تدريب هذا المولد؟ هذا هو السؤال الذي تناولته GAN.

قبل الحديث عن GAN، سنناقش المميز discriminator.

المميز

يأخذ المُميِّز discriminator صورة رقمية ويصنف ما إذا كانت الصورة حقيقية (1) أم لا (0).

إذا كانت الصورة المدخلة من قاعدة بيانات MNIST، فيجب على المُميِّز تصنيفها على أنها حقيقية real.

إذا كانت الصورة المدخلة من المولد، فيجب على المُميز تصنيفها على أنها مزيفة fake.

المُميِّز عبارة عن شبكة عصبية بسيطة متصلة بالكامل بطبقة واحدة مخفية مع تنشيط leaky ReLU.

```
discriminator = Sequential([
    Dense(128, input_shape=(784,)),
    LeakyReLU(alpha=0.01),
    Dense(1),
    Activation('sigmoid')
], name='discriminator')

discriminator.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 128)               100480
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 129
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 100,609
Trainable params: 100,609
Non-trainable params: 0
_________________________________________________________________
```

التنشيط الأخير هو sigmoid ليخبرنا باحتمالية ما إذا كانت الصورة المدخلة حقيقية أم لا.

نقوم بتدريب المُميِّز باستخدام كل من صور MNIST والصور التي تم إنشاؤها بواسطة المولد.

GAN

نقوم بتوصيل المولد والمميز لإنتاج GAN.

يأخذ العينة الكامنة، وينتج المولد الموجود داخل GAN صورة رقمية يصنفها المُميِّز داخل GAN على أنها حقيقية أو مزيفة.

إذا كانت الصورة الرقمية التي تم إنشاؤها واقعية جداً، فإن المُميزفي GAN يصنفها على أنها حقيقية، وهو ما نريد تحقيقه.

لقد قمنا بتعيين المُميِّز داخل GAN غير قابلة للتدريب not-trainable، لذا فهي تقوم فقط بتقييم جودة الصورة التي تم إنشاؤها. تكون التسمية دائمًا 1 (حقيقي) بحيث إذا فشل المولدفي إنتاج صورة رقمية واقعية، تصبح تكلفتها مرتفعة، وعندما يحدث الانتشار الخلفي في GAN، يتم تحديث الأوزان في شبكة المولد.

```
# maintain the same shared weights with the generator and
the discriminator.
gan = Sequential([
    generator,
    discriminator
])

gan.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
generator (Sequential)       (None, 784)               114064
_________________________________________________________________
discriminator (Sequential)   (None, 1)                 100609
=================================================================
Total params: 214,673
Trainable params: 214,673
Non-trainable params: 0
_________________________________________________________________
```

كما ترون، تستخدم GAN داخليًا نفس نماذج المولد والمميز. تحتفظ GAN بنفس الأوزان المشتركة مع المولد والمميز. لذلك، فإن تدريب GAN يعمل أيضًا على تدريب المولد. ومع ذلك، لا نريد أن يتأثر المُميِّز أثناء تدريب GAN.

نقوم بتدريب المُميِّز وGAN بدورهم ونكرر التدريب عدة مرات حتى يتم تدريبهما جيداً.

أثناء تدريب GAN، يجب أن يقوم الانتشار الخلفي بتحديث أوزان المولد وليس المُميِّز.

على هذا النحو، نحن بحاجة إلى طريقة لجعل المُميِّز قابلاً للتدريب وغير قابل للتدريب.

```
def make_trainable(model, trainable):
    for layer in model.layers:
        layer.trainable = trainable
```

```
make_trainable(discriminator, False)
discriminator.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 128)               100480
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 129
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 100,609
Trainable params: 0
```

```
Non-trainable params: 100,609
_________________________________________________________________
make_trainable(discriminator, True)
discriminator.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 128)               100480
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 129
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 100,609
Trainable params: 100,609
Non-trainable params: 0
_________________________________________________________________
```

تجمع الدالة أدناه كل ما ناقشناه حتى الآن لبناء نماذج المولد والمميز وGAN وتجميعها أيضًا للتدريب.

```
def make_simple_GAN(sample_size,
                    g_hidden_size,
                    d_hidden_size,
                    leaky_alpha,
                    g_learning_rate,
                    d_learning_rate):
    K.clear_session()

    generator = Sequential([
        Dense(g_hidden_size, input_shape=(sample_size,)),
        LeakyReLU(alpha=leaky_alpha),
        Dense(784),
        Activation('tanh')
    ], name='generator')

    discriminator = Sequential([
        Dense(d_hidden_size, input_shape=(784,)),
        LeakyReLU(alpha=leaky_alpha),
        Dense(1),
        Activation('sigmoid')
    ], name='discriminator')

    gan = Sequential([
        generator,
        discriminator
    ])
```

```
discriminator.compile(optimizer=Adam(lr=d_learning_rate),
loss='binary_crossentropy')
    gan.compile(optimizer=Adam(lr=g_learning_rate),
loss='binary_crossentropy')

    return gan, generator, discriminator
```

تدريب GAN

المعالجة المسبقة

نحن بحاجة إلى تسطيح flatten بيانات الصورة الرقمية حيث تتوقع طبقة الإدخال المتصلة بالكامل ذلك. أيضًا، نظرًا لأن المولد يستخدم تنشيط tanhفي طبقة الإخراج، فإننا نقوم بقياس جميع صور MNIST للحصول على قيم تتراوح بين ـ1 و1.

```
def preprocess(x):
    x = x.reshape(-1, 784) # 784=28*28
    x = np.float64(x)
    x = (x / 255 - 0.5) * 2
    x = np.clip(x, -1, 1)
    return x
```

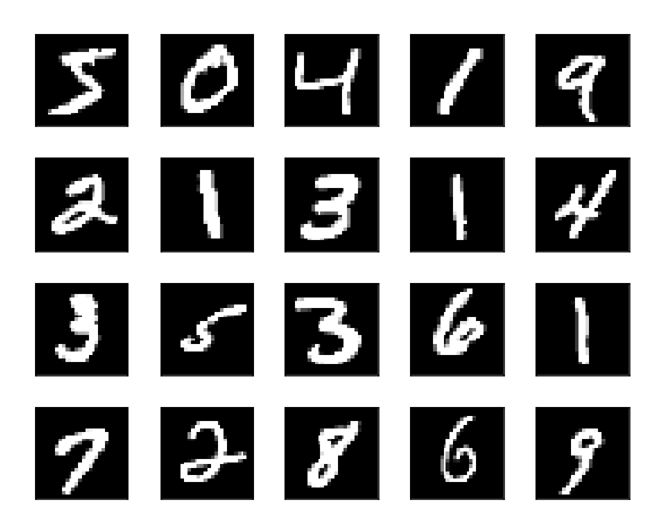
```
X_train_real = preprocess(X_train)
X_test_real  = preprocess(X_test)
```

إزالة المعالجة

نحتاج أيضًا إلى دالة لعكس المعالجة المسبقة حتى نتمكن من عرض الصور التي تم إنشاؤها.

```
def deprocess(x):
    x = (x / 2 + 1) * 255
    x = np.clip(x, 0, 255)
    x = np.uint8(x)
    x = x.reshape(28, 28)
    return x
```

```
plt.figure(figsize=(5, 4))
for i in range(20):
    img = deprocess(X_train_real[i])
    plt.subplot(4, 5, i+1)
    plt.imshow(img, cmap='gray')
    plt.xticks([])
    plt.yticks([])
plt.tight_layout()
plt.show()
```



التسميات

التسميات labels هي 1 (حقيقية real) أو 0 (مزيفة fake)في شكل ثنائي الأبعاد.

```
def make_labels(size):
    return np.ones([size, 1]), np.zeros([size, 1])
```

فيما يلي 10 مجموعات من قيم التسميات الحقيقية والمزيفة.

```
y_real_10, y_fake_10 = make_labels(10)

y_real_10, y_fake_10
```

```
(array([[ 1.],
        [ 1.],
        [ 1.],
        [ 1.],
        [ 1.],
        [ 1.],
        [ 1.],
        [ 1.],
        [ 1.],
        [ 1.]]), array([[ 0.],
        [ 0.],
        [ 0.],
        [ 0.],
        [ 0.],
        [ 0.],
        [ 0.],
        [ 0.],
        [ 0.],
        [ 0.]]))
```

لاحقًا، نقوم بإعداد التسميات للتدريب والتقييم باستخدام حجم دفعة التدريب وحجم الاختبار.

تنعيم التسميات

نقطة أخيرة قبل أن نبدأ التدريب هي تنعيم التسمية label smoothing مما يجعل المميز يعمم بشكل أفضل.

بالنسبة للصور الرقمية الحقيقية، تكون التسميات كلها بالرقم 1. ومع ذلك، عندما ندرب أداة المميز، نستخدم قيمة أصغر قليلاً من 1 مع الصور الرقمية الحقيقية. وبخلاف ذلك، قد يبالغ المُميِّزفي بيانات التدريب ويرفض أي شيء آخر يختلف قليلًا عن صور التدريب.

حلقة التدريب

نكرر ما يلي لجعل كل من المميز والمولد أفضل وأفضل:

- إعداد مجموعة من الصور الحقيقية.
- إعداد مجموعة من الصور المزيفة التي تم إنشاؤها بواسطة المولد باستخدام العينات الكامنة.
- جعل المميز قابلة للتدريب.
- تدريب المميز على تصنيف الصور الحقيقية والمزيفة.
- جعل المميز غير قابل للتدريب.
- تدريب المولد عبر GAN

عند تدريب المولد عبر GAN، تكون التسميات المتوقعة كلها 1 (حقيقية).في البداية، لا ينتج المولد صورًا واقعية جدًا، لذا يصنفها المُميِّز على أنها 0 (زائفة)، مما يتسببفي قيام الانتشار الخلفي بضبط الأوزان داخل المولد. لا يتأثر المُميِّز لأننا جعلناه غير قابل للتدريبفي هذه الخطوة.

```
# hyperparameters
sample_size     = 100     # latent sample size (i.e., 100
random numbers)
g_hidden_size   = 128
d_hidden_size   = 128
leaky_alpha     = 0.01
g_learning_rate = 0.0001  # learning rate for the generator
d_learning_rate = 0.001   # learning rate for the
discriminator
epochs          = 100
batch_size      = 64      # train batch size
eval_size       = 16      # evaluate size
smooth          = 0.1

# labels for the batch size and the test size
y_train_real, y_train_fake = make_labels(batch_size)
y_eval_real,  y_eval_fake  = make_labels(eval_size)

# create a GAN, a generator and a discriminator
```

```
gan, generator, discriminator = make_simple_GAN(
    sample_size,
    g_hidden_size,
    d_hidden_size,
    leaky_alpha,
    g_learning_rate,
    d_learning_rate)

losses = []
for e in range(epochs):
    for i in range(len(X_train_real)//batch_size):
        # real MNIST digit images
        X_batch_real =
X_train_real[i*batch_size:(i+1)*batch_size]

        # latent samples and the generated digit images
        latent_samples = make_latent_samples(batch_size,
sample_size)
        X_batch_fake =
generator.predict_on_batch(latent_samples)

        # train the discriminator to detect real and fake
images
        make_trainable(discriminator, True)
        discriminator.train_on_batch(X_batch_real,
y_train_real * (1 - smooth))
        discriminator.train_on_batch(X_batch_fake,
y_train_fake)

        # train the generator via GAN
        make_trainable(discriminator, False)
        gan.train_on_batch(latent_samples, y_train_real)

    # evaluate
    X_eval_real =
X_test_real[np.random.choice(len(X_test_real), eval_size,
replace=False)]

    latent_samples = make_latent_samples(eval_size,
sample_size)
    X_eval_fake = generator.predict_on_batch(latent_samples)

    d_loss  = discriminator.test_on_batch(X_eval_real,
y_eval_real)
    d_loss += discriminator.test_on_batch(X_eval_fake,
y_eval_fake)
    g_loss  = gan.test_on_batch(latent_samples, y_eval_real)
# we want the fake to be realistic!

    losses.append((d_loss, g_loss))
```

```
    print("Epoch: {:>3}/{} Discriminator Loss: {:>6.4f}
Generator Loss: {:>6.4f}".format(
        e+1, epochs, d_loss, g_loss))
```

```
Epoch:   1/100 Discriminator Loss: 0.9538 Generator Loss: 5.5816
Epoch:   2/100 Discriminator Loss: 0.1899 Generator Loss: 2.1410
Epoch:   3/100 Discriminator Loss: 0.3655 Generator Loss: 1.4118
Epoch:   4/100 Discriminator Loss: 0.1344 Generator Loss: 3.0701
Epoch:   5/100 Discriminator Loss: 0.3473 Generator Loss: 2.3738
Epoch:   6/100 Discriminator Loss: 0.5153 Generator Loss: 3.7222
Epoch:   7/100 Discriminator Loss: 0.5159 Generator Loss: 3.5699
Epoch:   8/100 Discriminator Loss: 0.7343 Generator Loss: 2.1149
Epoch:   9/100 Discriminator Loss: 0.4661 Generator Loss: 2.3607
Epoch:  10/100 Discriminator Loss: 0.2888 Generator Loss: 2.4522
Epoch:  11/100 Discriminator Loss: 0.4095 Generator Loss: 1.5612
Epoch:  12/100 Discriminator Loss: 0.5406 Generator Loss: 2.7845
Epoch:  13/100 Discriminator Loss: 0.3287 Generator Loss: 2.6246
Epoch:  14/100 Discriminator Loss: 0.2806 Generator Loss: 2.9350
Epoch:  15/100 Discriminator Loss: 0.6982 Generator Loss: 3.5069
Epoch:  16/100 Discriminator Loss: 0.4339 Generator Loss: 3.3039
Epoch:  17/100 Discriminator Loss: 0.7092 Generator Loss: 1.9226
Epoch:  18/100 Discriminator Loss: 0.8024 Generator Loss: 4.9868
Epoch:  19/100 Discriminator Loss: 0.2961 Generator Loss: 2.8417
Epoch:  20/100 Discriminator Loss: 0.5851 Generator Loss: 3.4906
Epoch:  21/100 Discriminator Loss: 0.3387 Generator Loss: 1.9244
Epoch:  22/100 Discriminator Loss: 0.4378 Generator Loss: 3.1012
Epoch:  23/100 Discriminator Loss: 0.2871 Generator Loss: 2.0432
Epoch:  24/100 Discriminator Loss: 0.3734 Generator Loss: 2.6555
Epoch:  25/100 Discriminator Loss: 0.7119 Generator Loss: 2.8028
Epoch:  26/100 Discriminator Loss: 0.1978 Generator Loss: 2.8457
Epoch:  27/100 Discriminator Loss: 0.5232 Generator Loss: 2.5416
Epoch:  28/100 Discriminator Loss: 0.2756 Generator Loss: 2.2270
Epoch:  29/100 Discriminator Loss: 0.3289 Generator Loss: 3.0124
Epoch:  30/100 Discriminator Loss: 0.5604 Generator Loss: 3.3040
Epoch:  31/100 Discriminator Loss: 0.8915 Generator Loss: 3.2336
Epoch:  32/100 Discriminator Loss: 0.6021 Generator Loss: 1.9195
Epoch:  33/100 Discriminator Loss: 0.4144 Generator Loss: 3.0869
Epoch:  34/100 Discriminator Loss: 0.6223 Generator Loss: 2.1233
Epoch:  35/100 Discriminator Loss: 0.2667 Generator Loss: 2.5386
Epoch:  36/100 Discriminator Loss: 0.3951 Generator Loss: 2.5837
Epoch:  37/100 Discriminator Loss: 0.6367 Generator Loss: 2.4578
Epoch:  38/100 Discriminator Loss: 0.4050 Generator Loss: 2.6965
Epoch:  39/100 Discriminator Loss: 0.3699 Generator Loss: 2.7588
Epoch:  40/100 Discriminator Loss: 0.6437 Generator Loss: 2.1038
Epoch:  41/100 Discriminator Loss: 0.2477 Generator Loss: 3.1998
Epoch:  42/100 Discriminator Loss: 0.3357 Generator Loss: 3.3893
Epoch:  43/100 Discriminator Loss: 0.8104 Generator Loss: 2.9031
Epoch:  44/100 Discriminator Loss: 0.5600 Generator Loss: 3.6187
Epoch:  45/100 Discriminator Loss: 0.4684 Generator Loss: 2.3988
Epoch:  46/100 Discriminator Loss: 0.2899 Generator Loss: 3.8236
Epoch:  47/100 Discriminator Loss: 0.2372 Generator Loss: 3.9306
Epoch:  48/100 Discriminator Loss: 0.5744 Generator Loss: 2.6210
Epoch:  49/100 Discriminator Loss: 0.5644 Generator Loss: 2.2713
Epoch:  50/100 Discriminator Loss: 0.9803 Generator Loss: 2.6462
Epoch:  51/100 Discriminator Loss: 0.5349 Generator Loss: 3.0719
Epoch:  52/100 Discriminator Loss: 0.8361 Generator Loss: 3.4607
Epoch:  53/100 Discriminator Loss: 0.4824 Generator Loss: 3.0189
Epoch:  54/100 Discriminator Loss: 0.6155 Generator Loss: 2.7298
Epoch:  55/100 Discriminator Loss: 0.6074 Generator Loss: 2.4785
Epoch:  56/100 Discriminator Loss: 0.6182 Generator Loss: 2.7999
```

```
Epoch:  57/100 Discriminator Loss: 0.8172 Generator Loss: 2.2989
Epoch:  58/100 Discriminator Loss: 0.6180 Generator Loss: 3.2786
Epoch:  59/100 Discriminator Loss: 0.8217 Generator Loss: 3.1931
Epoch:  60/100 Discriminator Loss: 0.6151 Generator Loss: 3.0382
Epoch:  61/100 Discriminator Loss: 0.8208 Generator Loss: 3.2423
Epoch:  62/100 Discriminator Loss: 0.7167 Generator Loss: 2.0826
Epoch:  63/100 Discriminator Loss: 0.7112 Generator Loss: 2.6495
Epoch:  64/100 Discriminator Loss: 0.5140 Generator Loss: 2.6749
Epoch:  65/100 Discriminator Loss: 0.8169 Generator Loss: 2.7637
Epoch:  66/100 Discriminator Loss: 0.5278 Generator Loss: 2.1983
Epoch:  67/100 Discriminator Loss: 0.7610 Generator Loss: 3.1669
Epoch:  68/100 Discriminator Loss: 0.5442 Generator Loss: 2.8738
Epoch:  69/100 Discriminator Loss: 0.8466 Generator Loss: 2.0486
Epoch:  70/100 Discriminator Loss: 0.6251 Generator Loss: 2.2485
Epoch:  71/100 Discriminator Loss: 0.6418 Generator Loss: 2.2814
Epoch:  72/100 Discriminator Loss: 0.4677 Generator Loss: 2.2908
Epoch:  73/100 Discriminator Loss: 0.6132 Generator Loss: 2.8723
Epoch:  74/100 Discriminator Loss: 0.7114 Generator Loss: 2.6701
Epoch:  75/100 Discriminator Loss: 1.0905 Generator Loss: 1.9237
Epoch:  76/100 Discriminator Loss: 0.9982 Generator Loss: 2.5257
Epoch:  77/100 Discriminator Loss: 0.7214 Generator Loss: 2.1899
Epoch:  78/100 Discriminator Loss: 0.7138 Generator Loss: 1.9392
Epoch:  79/100 Discriminator Loss: 0.9038 Generator Loss: 2.3197
Epoch:  80/100 Discriminator Loss: 1.0450 Generator Loss: 3.0043
Epoch:  81/100 Discriminator Loss: 0.5381 Generator Loss: 2.9071
Epoch:  82/100 Discriminator Loss: 0.5621 Generator Loss: 2.5895
Epoch:  83/100 Discriminator Loss: 0.8544 Generator Loss: 3.3824
Epoch:  84/100 Discriminator Loss: 0.8167 Generator Loss: 2.6601
Epoch:  85/100 Discriminator Loss: 0.7621 Generator Loss: 2.9904
Epoch:  86/100 Discriminator Loss: 0.8123 Generator Loss: 2.7157
Epoch:  87/100 Discriminator Loss: 0.5252 Generator Loss: 3.1781
Epoch:  88/100 Discriminator Loss: 0.9563 Generator Loss: 2.1756
Epoch:  89/100 Discriminator Loss: 1.0338 Generator Loss: 2.7354
Epoch:  90/100 Discriminator Loss: 0.5451 Generator Loss: 3.0826
Epoch:  91/100 Discriminator Loss: 0.9634 Generator Loss: 3.0830
Epoch:  92/100 Discriminator Loss: 0.7814 Generator Loss: 2.9515
Epoch:  93/100 Discriminator Loss: 0.8324 Generator Loss: 3.5539
Epoch:  94/100 Discriminator Loss: 1.8759 Generator Loss: 1.9687
Epoch:  95/100 Discriminator Loss: 0.9151 Generator Loss: 2.2101
Epoch:  96/100 Discriminator Loss: 0.9279 Generator Loss: 2.2720
Epoch:  97/100 Discriminator Loss: 0.8894 Generator Loss: 3.2808
Epoch:  98/100 Discriminator Loss: 0.9052 Generator Loss: 2.5838
Epoch:  99/100 Discriminator Loss: 0.6043 Generator Loss: 2.9611
Epoch: 100/100 Discriminator Loss: 0.5693 Generator Loss: 2.3100
```

استقرار GAN

وكما تبين، فإن تدريب GAN أمر صعب للغاية، وهناك العديد من الحيل والاستدلالات المطلوبة. وذلك لأن المُميز والمولد لا يتعاونان ويتعلمان بشكل فردي التنبؤ بشكل أفضل.

على سبيل المثال، قد يتعلم المولد خداع المميز بالقمامة. من الناحية المثالية، يجب أن يتعلم المُميِّزفي وقت أبكر من المولد حتى يتمكن من تصنيف الصور بدقة.

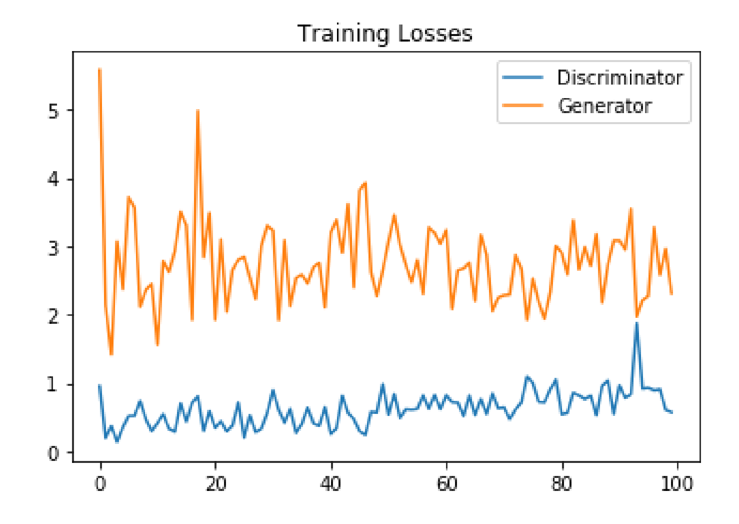
لذلك، استخدمت معدلات تعلم مختلفة للمولد والمميز. كنت أرغب في إبطاء تعلم المولد حتى يتعلم المُميِّز التصنيف جيداً.

لست متأكداً بنسبة 100% مما إذا كانت هذه استراتيجية جيدة للاستخدام بشكل عام ولكن يبدو أنها ناجحةفي هذا المشروع.

كلما تعلم المولد أكثر وتناقص الخطأ، زاد خطأ المُميِّز. أرى نوعًا من التوازن حوالي 80ـ90 فترة.

```
losses = np.array(losses)

fig, ax = plt.subplots()
plt.plot(losses.T[0], label='Discriminator')
plt.plot(losses.T[1], label='Generator')
plt.title("Training Losses")
plt.legend()
plt.show()
```

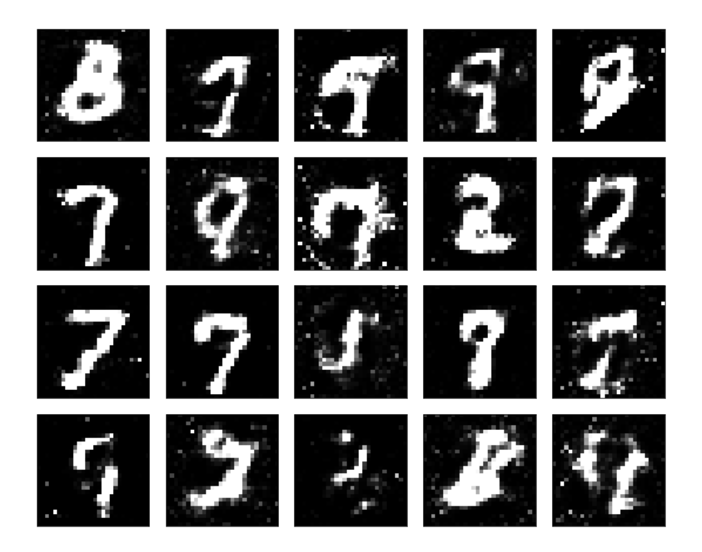


اختبار المولد

نقوم الآن بإنشاء بعض الصور الرقمية باستخدام المولد المدرب.

```
latent_samples = make_latent_samples(20, sample_size)
generated_digits = generator.predict(latent_samples)

plt.figure(figsize=(10, 8))
for i in range(20):
    img = deprocess(generated_digits[i])
    plt.subplot(4, 5, i+1)
    plt.imshow(img, cmap='gray')
    plt.xticks([])
    plt.yticks([])
plt.tight_layout()
plt.show()
```



المصدر:

https://notebook.community/naokishibuya/deep-learning/python/gan_mnist

6) إنشاء صور الموضة باستخدام شبكات الخصومة التوليدية Fashion Image Generation using GANs

مقدمة

سوف تستكشف هذه المقالة Generative Adversarial Networks (GANs) وقدرتها الرائعة على توليد صور الموضة fashion image generation. لقد أحدثت شبكات GAN ثورةفي مجال النمذجة التوليدية generative modeling، حيث تقدم نهجًا مبتكرًا لإنشاء محتوى جديد من خلال التعلم التنافسي adversarial learning.

خلال هذا الدليل، سنأخذكفي رحلة آسرة، بدءًا من المفاهيم الأساسية لشبكات GAN والتعمق تدريجيًا في تعقيدات إنشاء صور الموضة. من خلال المشاريع العملية والتعليمات خطوة بخطوة، سنرشدك خلال بناء نموذج GAN الخاص بك وتدريبه باستخدام TensorFlow وKeras.

استعد لإطلاق العنان لإمكانات شبكات GAN وشاهد سحر الذكاء الاصطناعيفي عالم الموضة. سواء كنت ممارسًا متمرسًافي مجال الذكاء الاصطناعي أو متحمسًا للفضول، ستزودك دورة " GANS in Vogue" بالمهارات والمعرفة اللازمة لإنشاء تصميمات أزياء مذهلة ودفع حدود الفن التوليدي. دعونا نتعمقفي عالم شبكات GAN الرائعة ونطلق العنان للإبداع بداخلها!



فهم شبكات الخصومة التوليدية (GANs)

ما هي شبكات GAN؟

تتكون شبكات الخصومة التوليدية (GANs) من شبكتين عصبيتين: المولد generator والمميز discriminator. المولد مسؤول عن إنشاء عينات بيانات جديدة، بينما مهمة المميز هي التمييز بين البيانات الحقيقية real data والبيانات المزيفة fake data التي يولدها المولد. يتم تدريب الشبكتين في وقت واحد من خلال عملية تنافسية competitive process، حيث يقوم المولد بتحسين قدرته على إنشاء عينات واقعية بينما يصبح المميز أفضل في التعرف على الحقيقي من المزيف.

كيف تعمل شبكات GAN؟

تعتمد شبكات GAN على سيناريو يشبه اللعبة حيث يلعب المولد والمميز ضد بعضهما البعض. يحاول المولد إنشاء بيانات تشبه البيانات الحقيقية، بينما يهدف المميز إلى التمييز بين البيانات الحقيقية والمزيفة. يتعلم المولد كيفية إنشاء عينات أكثر واقعية من خلال عملية التدريب العدائية adversarial training process هذه.

المكونات الرئيسية لشبكات GAN

لبناء شبكة GAN، نحتاج إلى عدة مكونات أساسية:

- **المولد Generator:** شبكة عصبية تولد عينات بيانات جديدة.
- **المميز Discriminator:** شبكة عصبية تصنف البيانات على أنها حقيقية أو مزيفة.
- **المساحة الكامنة Latent Space:** مساحة متجهة عشوائية يستخدمها المولد كمدخل لإنتاج العينات.
- **حلقة التدريب Training Loop:** العملية التكرارية لتدريب المولد والمميزفي خطوات متناوبة.

دوال الخطأ في شبكات GAN

تعتمد عملية تدريب GAN على دوال خطأ loss functions محددة. يحاول المولد تقليل خطأ المولد، مما يشجعه على إنشاء بيانات أكثر واقعية.في الوقت نفسه، يهدف المُميِّز إلى تقليل خطأ المُميِّز، ليصبح أفضل في التمييز بين البيانات الحقيقية والمزيفة.

نظرة عامة على المشروع: إنشاء صور الموضة باستخدام شبكات GAN

هدف المشروع

في هذا المشروع، نهدف إلى بناء GAN لإنشاء صور أزياء جديدة تشبه تلك الموجودةفي مجموعة بيانات Fashion MNIST. يجب أن تلتقط الصور التي تم إنشاؤها السمات الأساسية لمختلف عناصر الموضة، مثل الفساتين والقمصان والسراويل والأحذية.



مجموعة البيانات: Fashion MNIST

سوف نستخدم مجموعة بيانات Fashion MNIST، وهي مجموعة بيانات مرجعية شائعة تحتوي على صور ذات تدرج رمادي لعناصر الموضة. تبلغ أبعاد كل صورة 28 × 28 بكسل، وهناك عشر فئات في المجمل.

تهيئة بيئة المشروع

للبدء، يجب علينا إعداد بيئة Python الخاصة بنا وتثبيت المكتبات اللازمة، بمافي ذلك TensorFlow وMatplotlib وTensorFlow Datasets.

بناء شبكة GAN

استيراد التبعيات والبيانات

للبدء، يجب علينا تثبيت واستيراد المكتبات اللازمة وتحميل مجموعة بيانات Fashion MNIST التي تحتوي على مجموعة من صور الموضة. سوف نستخدم مجموعة البيانات هذه لتدريب نموذج الذكاء الاصطناعي الخاص بنا لإنشاء صور أزياء جديدة.

```
# Install required packages (only need to do this once)
!pip install tensorflow tensorflow-gpu matplotlib
tensorflow-datasets ipywidgets
!pip list

# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten,
Reshape, LeakyReLU, Dropout, UpSampling2D
import tensorflow_datasets as tfds
from matplotlib import pyplot as plt

# Configure TensorFlow to use GPU for faster computation
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)

# Load the Fashion MNIST dataset
ds = tfds.load('fashion_mnist', split='train')
```

تصور البيانات وبناء مجموعة البيانات

بعد ذلك، سنقوم بتصور عينة من الصور من مجموعة بيانات Fashion MNIST وإعداد مسار البيانات. سنقوم بإجراء تحويلات للبيانات data transformations وإنشاء مجموعات من الصور لتدريب شبكة GAN.

```
# Data Transformation: Scale and Vizualize Images
import numpy as np

# Setup data iterator
dataiterator = ds.as_numpy_iterator()

# Visualize some images from the dataset
fig, ax = plt.subplots(ncols=4, figsize=(20, 20))

# Loop four times and get images
for idx in range(4):
    # Grab an image and its label
    sample = dataiterator.next()
```

```
    image = np.squeeze(sample['image'])  # Remove the 
single-dimensional entries
    label = sample['label']

    # Plot the image using a specific subplot
    ax[idx].imshow(image)
    ax[idx].title.set_text(label)

# Data Preprocessing: Scale and Batch the Images
def scale_images(data):
    # Scale the pixel values of the images between 0 and 1
    image = data['image']
    return image / 255.0

# Reload the dataset
ds = tfds.load('fashion_mnist', split='train')

# Apply the scale_images preprocessing step to the dataset
ds = ds.map(scale_images)

# Cache the dataset for faster processing during training
ds = ds.cache()

# Shuffle the dataset to add randomness to the training 
process
ds = ds.shuffle(60000)

# Batch the dataset into smaller groups (128 images per 
batch)
ds = ds.batch(128)

# Prefetch the dataset to improve performance during 
training
ds = ds.prefetch(64)

# Check the shape of a batch of images
ds.as_numpy_iterator().next().shape
```

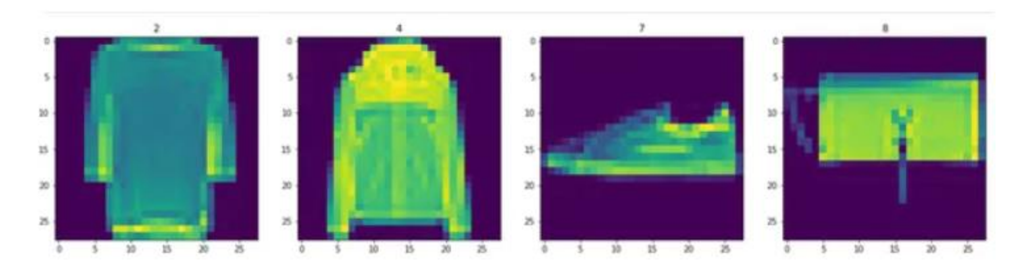
في هذه الخطوة، نقوم أولاً بتصور أربع صور أزياء عشوائية من مجموعة البيانات باستخدام مكتبة matplotlib. يساعدنا هذا على فهم شكل الصور وما نريد أن يتعلمه نموذج الذكاء الاصطناعي الخاص بنا.

بعد تصور الصور، ننتقل إلى المعالجة المسبقة للبيانات data preprocessing. نقوم بقياس قيم البكسل للصور بين 0 و1، مما يساعد نموذج الذكاء الاصطناعي على التعلم بشكل أفضل. تخيل أن تقوم بتحجيم scaling سطوع الصور لتكون مناسبة للتعلم.

بعد ذلك، نقوم بتجميع الصورفي مجموعات مكونة من 128 صورة (دفعة batch) لتدريب نموذج الذكاء الاصطناعي الخاص بنا. فكرفي الدفعات على أنها تقسيم مهمة كبيرة إلى أجزاء أصغر يمكن التحكم فيها.

نقوم أيضًا بتبديل مجموعة البيانات عشوائيًا لإضافة بعض العشوائية حتى لا يتعرف نموذج الذكاء الاصطناعي على الصور بترتيب ثابت.

وأخيرًا، نقوم بإحضار البيانات مسبقًا لإعدادها لعملية التعلم الخاصة بنموذج الذكاء الاصطناعي، مما يجعلها تعمل بشكل أسرع وأكثر كفاءة.



في نهاية هذه الخطوة، قمنا بتصور بعض صور الأزياء، وتم إعداد وتنظيم مجموعة البيانات الخاصة بنا لتدريب نموذج الذكاء الاصطناعي. نحن الآن جاهزون للانتقال إلى الخطوة التالية، حيث سنقوم ببناء الشبكة العصبية لتوليد صور أزياء جديدة.

بناء المولد

يُعد المولد أمرًا بالغ الأهمية لـ GAN، حيث يقوم بإنشاء صور أزياء جديدة. سنقوم بتصميم المولد باستخدام واجهة برمجة التطبيقات التسلسلية Sequential API الخاصة بـ TensorFlow، والتي تتضمن طبقات مثل Dense وLeakyReLU وReshape وConv2DTranspose.

```
# Import the Sequential API for building models
from tensorflow.keras.models import Sequential

# Import the layers required for the neural network
from tensorflow.keras.layers import (
    Conv2D, Dense, Flatten, Reshape, LeakyReLU, Dropout,
UpSampling2D
)
```

```
def build_generator():
    model = Sequential()

    # First layer takes random noise and reshapes it to
7x7x128
```

```
    # This is the beginning of the generated image
    model.add(Dense(7 * 7 * 128, input_dim=128))
    model.add(LeakyReLU(0.2))
    model.add(Reshape((7, 7, 128)))

    # Upsampling block 1
    model.add(UpSampling2D())
    model.add(Conv2D(128, 5, padding='same'))
    model.add(LeakyReLU(0.2))

    # Upsampling block 2
    model.add(UpSampling2D())
    model.add(Conv2D(128, 5, padding='same'))
    model.add(LeakyReLU(0.2))

    # Convolutional block 1
    model.add(Conv2D(128, 4, padding='same'))
    model.add(LeakyReLU(0.2))

    # Convolutional block 2
    model.add(Conv2D(128, 4, padding='same'))
    model.add(LeakyReLU(0.2))

    # Convolutional layer to get to one channel
    model.add(Conv2D(1, 4, padding='same',
activation='sigmoid'))

    return model

# Build the generator model
generator = build_generator()
# Display the model summary
generator.summary()
```

المولد عبارة عن شبكة عصبية عميقة مسؤولة عن إنشاء صور أزياء مزيفة. فهو يأخذ ضوضاء عشوائية كمدخل، ويكون إخراجه عبارة عن صورة ذات تدرج رمادي مقاس 28 × 28 تبدو كعنصر أزياء. الهدف هو تعلم كيفية إنشاء صور تشبه عناصر الموضة الحقيقية.

يتكون النموذج من عدة طبقات:

1. **Dense Layer**: الطبقة الأولى تأخذ ضوضاء عشوائية random noise بحجم 128 وتعيد تشكيلها إلى موتر 7x7x128. يؤدي هذا إلى إنشاء المعمارية الأولية للصورة التي تم إنشاؤها.
2. **Upsampling Blocks**: تعمل هذه الكتل على زيادة دقة الصورة تدريجيًا باستخدام طبقة UpSampling2D، تليها طبقة تلافيفية وتنشيط LeakyReLU. تعمل طبقة Upsampling2D على مضاعفة دقة الصورة على كلا البعدين.

3. **Convolutional Blocks**: تعمل هذه الكتل على تحسين الصورة التي تم إنشاؤها. وهي تتكون من طبقات تلافيفية convolutional layers مع عمليات تنشيط LeakyReLU.
4. **Convolutional Layer**: الطبقة التلافيفية النهائية تقلل القنوات إلى قناة واحدة، مما يؤدي بشكل فعال إلى إنشاء صورة الإخراج مع التنشيط sigmoid لقياس قيم البكسل بين 0 و1.

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 6272)              809088

 leaky_re_lu_26 (LeakyReLU)  (None, 6272)              0

 reshape_9 (Reshape)         (None, 7, 7, 128)         0

 up_sampling2d_12 (UpSamplin  (None, 14, 14, 128)      0
 g2D)

 conv2d_19 (Conv2D)          (None, 14, 14, 128)       409728

 leaky_re_lu_27 (LeakyReLU)  (None, 14, 14, 128)       0

 up_sampling2d_13 (UpSamplin  (None, 28, 28, 128)      0
 g2D)

 conv2d_20 (Conv2D)          (None, 28, 28, 128)       409728

 leaky_re_lu_28 (LeakyReLU)  (None, 28, 28, 128)       0

 conv2d_21 (Conv2D)          (None, 28, 28, 128)       262272

 leaky_re_lu_29 (LeakyReLU)  (None, 28, 28, 128)       0

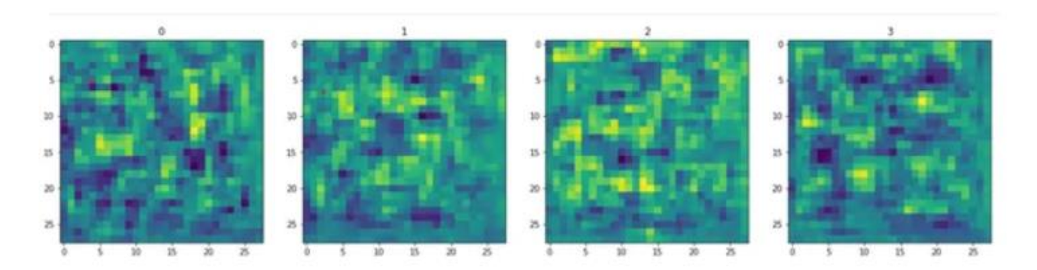
 conv2d_22 (Conv2D)          (None, 28, 28, 128)       262272

 leaky_re_lu_30 (LeakyReLU)  (None, 28, 28, 128)       0

 conv2d_23 (Conv2D)          (None, 28, 28, 1)         2049

=================================================================
Total params: 2,155,137
Trainable params: 2,155,137
Non-trainable params: 0
```

في نهاية هذه الخطوة، سيكون لدينا نموذج مولد قادر على إنتاج صور أزياء مزيفة. النموذج جاهز الآن للتدريب على الخطوات التالية من العملية.



بناء المميز

بدءًا من المفاهيم الأساسية لشبكات GAN والتعمق تدريجيًافي تعقيدات توليد صور الموضة. من خلال المشاريع العملية والتعليمات خطوة بخطوة، سنرشدك خلال بناء نموذج GAN الخاص بك وتدريبه باستخدام TensorFlow وKeras.

يلعب المميز دورًا حاسمًافي التمييز بين الصور الحقيقية والمزيفة. سنقوم بتصميم المميز باستخدام واجهة برمجة التطبيقات التسلسلية Sequential API الخاصة بـ TensorFlow، والتي تتضمن طبقات Conv2D وLeakyReLU وDropout وDense.

```
def build_discriminator():
    model = Sequential()

    # First Convolutional Block
    model.add(Conv2D(32, 5, input_shape=(28, 28, 1)))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.4))

    # Second Convolutional Block
    model.add(Conv2D(64, 5))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.4))

    # Third Convolutional Block
    model.add(Conv2D(128, 5))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.4))

    # Fourth Convolutional Block
    model.add(Conv2D(256, 5))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.4))

    # Flatten the output and pass it through a dense layer
    model.add(Flatten())
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))

    return model

# Build the discriminator model
discriminator = build_discriminator()
# Display the model summary
discriminator.summary()
```

يعد المُميِّز أيضًا شبكة عصبية عميقة لتصنيف ما إذا كانت الصورة المدخلة حقيقية أم مزيفة. يقوم بإدخال صورة ذات تدرج رمادي مقاس 28 × 28 ويخرج قيمة ثنائية (1 للحقيقي، 0 للمزيف).

يتكون النموذج من عدة طبقات:

1. **Convolutional Blocks:** تقوم هذه الكتل بمعالجة الصورة المدخلة بطبقات تلافيفية convolutional layers، تليها عمليات تنشيط LeakyReLU والطبقات المتسربة

dropout layers. تساعد الطبقات المتسربة على منع الضبط الزائد overfitting عن طريق إسقاط (حذف) بعض الخلايا العصبية بشكل عشوائي أثناء التدريب.

2. **Flatten and Dense Layers**: يتم تسوية مخرجات الكتلة التلافيفية الأخيرة إلى متجه أحادي الأبعاد ويتم تمريرها عبر طبقة كثيفة مع التنشيط sigmoid. يؤدي التنشيط sigmoid إلى سحق الإخراج بين 0 و1، مما يمثل احتمال أن تكون الصورة حقيقية.

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_32 (Conv2D)          (None, 24, 24, 32)        832

 leaky_re_lu_39 (LeakyReLU)  (None, 24, 24, 32)        0

 dropout_8 (Dropout)         (None, 24, 24, 32)        0

 conv2d_33 (Conv2D)          (None, 20, 20, 64)        51264

 leaky_re_lu_40 (LeakyReLU)  (None, 20, 20, 64)        0

 dropout_9 (Dropout)         (None, 20, 20, 64)        0

 conv2d_34 (Conv2D)          (None, 16, 16, 128)       204928

 leaky_re_lu_41 (LeakyReLU)  (None, 16, 16, 128)       0

 dropout_10 (Dropout)        (None, 16, 16, 128)       0

 conv2d_35 (Conv2D)          (None, 12, 12, 256)       819456

 leaky_re_lu_42 (LeakyReLU)  (None, 12, 12, 256)       0

 dropout_11 (Dropout)        (None, 12, 12, 256)       0

 flatten (Flatten)           (None, 36864)             0

 dropout_12 (Dropout)        (None, 36864)             0

 dense_11 (Dense)            (None, 1)                 36865

=================================================================
Total params: 1,113,345
Trainable params: 1,113,345
Non-trainable params: 0
_________________________________________________________________
```

في نهاية هذه الخطوة، سيكون لدينا نموذج مميز قادر على تصنيف ما إذا كانت الصورة المدخلة حقيقية أم مزيفة. النموذج جاهز الآن للدمج في معمارية GAN وتدريبه على الخطوات التالية.

بناء حلقة التدريب

إعداد الاخطاء والمحسن

قبل بناء حلقة التدريب، نحتاج إلى تحديد دوال الخطأ والمحسنات التي سيتم استخدامها لتدريب كل من المولد والمميز.

```
# Import the Adam optimizer and Binary Cross Entropy loss
function
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy

# Define the optimizers for the generator and discriminator
g_opt = Adam(learning_rate=0.0001)  # Generator optimizer
```

```
d_opt = Adam(learning_rate=0.00001)  # Discriminator
optimizer

# Define the loss functions for the generator and
discriminator
g_loss = BinaryCrossentropy()  # Generator loss function
d_loss = BinaryCrossentropy()  # Discriminator loss function
```

نحن نستخدم مُحسِّن Adam لكل من المولد والمميز. Adam عبارة عن خوارزمية تحسين فعالة تعمل على تكييف معدل التعلم learning rate أثناء التدريب.

بالنسبة لدوال الخطأ، نحن نستخدم الانتروبيا المتقاطعة الثنائية Binary Cross Entropy. تُستخدم دالة الخطأ هذه بشكل شائعفي مشاكل التصنيف الثنائي، وهي مناسبة لمهمة التصنيف الثنائي الخاصة بأداة المميز (الحقيقية مقابل المزيفة).

بناء نموذج فرعي

بعد ذلك، سنقوم ببناء نموذج فرعي subclassed model يجمع بين نماذج المولد والمميزفي نموذج GAN واحد. سيقوم هذا النموذج الفرعي بتدريب GAN أثناء حلقة التدريب.

```
from tensorflow.keras.models import Model

class FashionGAN(Model):
    def __init__(self, generator, discriminator, *args,
**kwargs):
        # Pass through args and kwargs to the base class
        super().__init__(*args, **kwargs)

        # Create attributes for generator and discriminator
models
        self.generator = generator
        self.discriminator = discriminator

    def compile(self, g_opt, d_opt, g_loss, d_loss, *args,
**kwargs):
        # Compile with the base class
        super().compile(*args, **kwargs)

        # Create attributes for optimizers and loss
functions
        self.g_opt = g_opt
        self.d_opt = d_opt
        self.g_loss = g_loss
        self.d_loss = d_loss

    def train_step(self, batch):
        # Get the data for real images
        real_images = batch
```

```
        # Generate fake images using the generator with
random noise as input
        fake_images = self.generator(tf.random.normal((128,
128, 1)), training=False)

        # Train the discriminator
        with tf.GradientTape() as d_tape:
            # Pass real and fake images through the
discriminator model
            yhat_real = self.discriminator(real_images,
training=True)
            yhat_fake = self.discriminator(fake_images,
training=True)
            yhat_realfake = tf.concat([yhat_real,
yhat_fake], axis=0)

            # Create labels for real and fake images
            y_realfake =
tf.concat([tf.zeros_like(yhat_real),
tf.ones_like(yhat_fake)], axis=0)

            # Add some noise to the true outputs to make
training more robust
            noise_real = 0.15 *
tf.random.uniform(tf.shape(yhat_real))
            noise_fake = -0.15 *
tf.random.uniform(tf.shape(yhat_fake))
            y_realfake += tf.concat([noise_real,
noise_fake], axis=0)

            # Calculate the total discriminator loss
            total_d_loss = self.d_loss(y_realfake,
yhat_realfake)

        # Apply backpropagation and update discriminator
weights
        dgrad = d_tape.gradient(total_d_loss,
self.discriminator.trainable_variables)
        self.d_opt.apply_gradients(zip(dgrad,
self.discriminator.trainable_variables))

        # Train the generator
        with tf.GradientTape() as g_tape:
            # Generate new images using the generator with
random noise as input
            gen_images =
self.generator(tf.random.normal((128, 128, 1)),
training=True)
```

```
            # Create the predicted labels (should be close
to 1 as they are fake images)
            predicted_labels =
self.discriminator(gen_images, training=False)

            # Calculate the total generator loss (tricking
the discriminator to classify the fake images as real)
            total_g_loss =
self.g_loss(tf.zeros_like(predicted_labels),
predicted_labels)

        # Apply backpropagation and update generator weights
        ggrad = g_tape.gradient(total_g_loss,
self.generator.trainable_variables)
        self.g_opt.apply_gradients(zip(ggrad,
self.generator.trainable_variables))

        return {"d_loss": total_d_loss, "g_loss":
total_g_loss}

# Create an instance of the FashionGAN model
fashgan = FashionGAN(generator, discriminator)

# Compile the model with the optimizers and loss functions
fashgan.compile(g_opt, d_opt, g_loss, d_loss)
```

- نقوم بإنشاء نموذج FashionGAN ذو فئة فرعية (subclassed FashionGAN) يمتد إلى فئة tf.keras.models.Model. سيتعامل هذا النموذج الفرعي مع عملية التدريب لشبكة GAN.
- في طريقة Train_step، نحدد حلقة التدريب لشبكة GAN:
  - نحصل أولاً على صور أصلية من الدفعة وننشئ صورًا مزيفة باستخدام نموذج المولد مع ضوضاء عشوائية كمدخل.
  - ثم نقوم بتدريب المميز:
    - نستخدم شريطًا متدرجًا gradient tape لحساب خطأ المميز فيما يتعلق بالصور الحقيقية والمزيفة. الهدف هو جعل المميز تصنف الصور الأصلية على أنها 1 والصور المزيفة على أنها 0.
    - نضيف بعض الضوضاء إلى المخرجات الحقيقية لجعل التدريب أكثر قوة وأقل عرضة للضبط الزائد.
    - يتم حساب إجمالي خطأ المميز على أنها إنتروبيا متقاطعة ثنائية بين التسميات المتوقعة والمستهدفة.

- نحن نطبق الانتشار الخلفي backpropagation لتحديث أوزان المُميِّز بناءً على الخطأ المحسوب.

  ○ بعد ذلك نقوم بتدريب المولد:

- نقوم بإنشاء صور مزيفة جديدة باستخدام المولد مع ضوضاء عشوائية كمدخل.
- نحن نحسب إجمالي خطأ المولد على أنها إنتروبيا متقاطعة ثنائية بين التسميات المتوقعة (الصور التي تم إنشاؤها) والتسميات المستهدفة (0، تمثل صورًا مزيفة).
- يهدف المولد إلى "خداع" المُميِّز من خلال توليد صور يصنفها المُميِّز على أنها حقيقية (مع علامة قريبة من 1).
- نحن نطبق الانتشار الخلفي لتحديث أوزان المولد بناءً على الخطأ المحسوب.

  ○ وأخيرا، نعيد إجمالي الاخطاء للمميز والمولد خلال هذه الخطوة التدريبية.

أصبح نموذج FashionGAN جاهزًا الآن للتدريب باستخدام مجموعة بيانات التدريب في الخطوة التالية.

بناء رد الاتصال

عمليات رد الاتصال Callbacks في TensorFlow هي دوال يمكن تنفيذها أثناء التدريب في نقاط محددة، مثل نهاية فترة epoch ما. سنقوم بإنشاء رد اتصال مخصص يسمى ModelMonitor لإنشاء الصور وحفظها في نهاية كل فترة لمراقبة تقدم شبكة GAN.

```
import os
from tensorflow.keras.preprocessing.image import
array_to_img
from tensorflow.keras.callbacks import Callback

class ModelMonitor(Callback):
    def __init__(self, num_img=3, latent_dim=128):
        self.num_img = num_img
        self.latent_dim = latent_dim

    def on_epoch_end(self, epoch, logs=None):
        # Generate random latent vectors as input to the
generator
        random_latent_vectors =
tf.random.uniform((self.num_img, self.latent_dim, 1))
        # Generate fake images using the generator
        generated_images =
self.model.generator(random_latent_vectors)
        generated_images *= 255
        generated_images.numpy()
        for i in range(self.num_img):
            # Save the generated images to disk
```

```
            img = array_to_img(generated_images[i])
            img.save(os.path.join('images',
f'generated_img_{epoch}_{i}.png'))
```

- يأخذ رد الاتصال ModelMonitor وسيطتين: num_img، الذي يحدد عدد الصور التي سيتم إنشاؤها وحفظهافي نهاية كل فترة، وlatent_dim، وهو بُعد متجه الضوضاء العشوائي random noise vector المستخدم كمدخل للمولد.
- أثناء طريقة on_epoch_end، يقوم رد الاتصال بإنشاء num_img متجهات كامنة عشوائية وتمريرها كمدخلات إلى المولد. يقوم المولد بعد ذلك بإنشاء صور مزيفة بناءً على هذه المتجهات العشوائية.
- يتم تحجيم الصور التي تم إنشاؤها إلى نطاق 0ـ255 ويتم حفظها كملفات PNGفي دليل "images". تتضمن أسماء الملفات رقم الفترة لتتبع التقدم مع مرور الوقت.

تدريب GAN

الآن بعد أن قمنا بإعداد نموذج GAN ورد الاتصال المخصص، يمكننا بدء عملية التدريب باستخدام طريقة fit. سنقوم بتدريب GAN لفترات كافية للسماح للمولد والمميز بالتقارب converge والتعلم من بعضهما البعض.

```
# Train the GAN model
hist = fashgan.fit(ds, epochs=20,
callbacks=[ModelMonitor()])
```

- نحن نستخدم طريقة fit لنموذج FashionGAN لتدريب GAN.
- قمنا بتعيين عدد الفترات على 20 (قد تحتاج إلى المزيد من الفترات للحصول على نتائج أفضل).
- نقوم بتمرير رد اتصال ModelMonitor لحفظ الصور التي تم إنشاؤهافي نهاية كل فترة.
- سيتم تكرار عملية التدريب على مجموعة البيانات، ولكل دفعة، سيتم تحديث أوزان نماذج المولد والمميز باستخدام حلقة التدريب المحددة مسبقًا.

يمكن أن تستغرق عملية التدريب بعض الوقت، اعتمادًا على أجهزتك وعدد الفترات. بعد التدريب، يمكننا مراجعة أداء شبكة GAN من خلال رسم اخطاء المميز والمولد. سيساعدنا هذا على فهم مدى جودة تدريب النماذج وما إذا كانت هناك أي علامة على التقارب convergence أو انهيار الوضع mode collapse. دعنا ننتقل إلى الخطوة التالية، وهي مراجعة أداء GAN.

مراجعة الأداء واختبار المولد

مراجعة الأداء

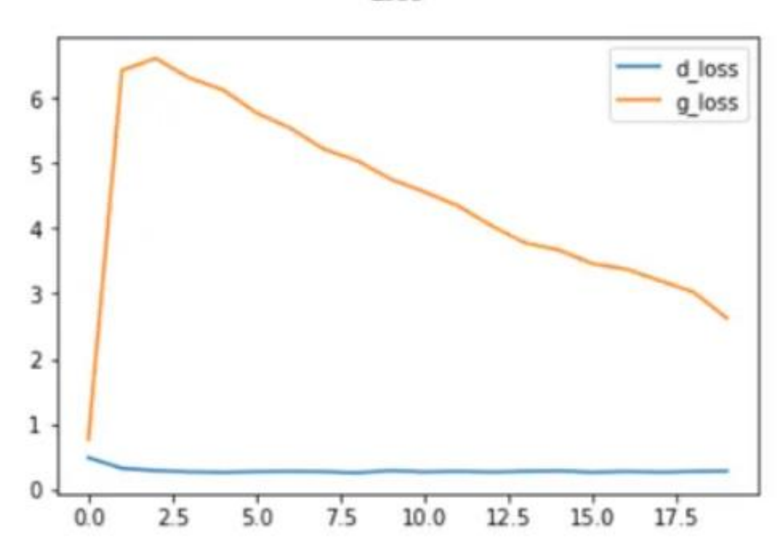
بعد تدريب GAN، يمكننا مراجعة أدائها من خلال رسم اخطاء المميز والمولد خلال فترات التدريب. سيساعدنا هذا على فهم مدى جودة تعلم GAN وما إذا كانت هناك أية مشكلات، مثل انهيار الوضع mode collapse أو التدريب غير المستقر unstable training.

```
import matplotlib.pyplot as plt

# Plot the discriminator and generator losses
plt.suptitle('Loss')
plt.plot(hist.history['d_loss'], label='d_loss')
plt.plot(hist.history['g_loss'], label='g_loss')
plt.legend()
plt.show()
```

- نحن نستخدم matplotlib لرسم اخطاء المميز والمولد خلال فترات التدريب.
- يمثل المحور السيني رقم الفترة، ويمثل المحور الصادي الاخطاء المقابلة.
- من المفترض أن تنخفض اخطاء المميز (d_loss) وخطأ المولد (g_loss) بشكل مثالي على مدار الفترات كما تتعلم GAN.





اختبار المولد

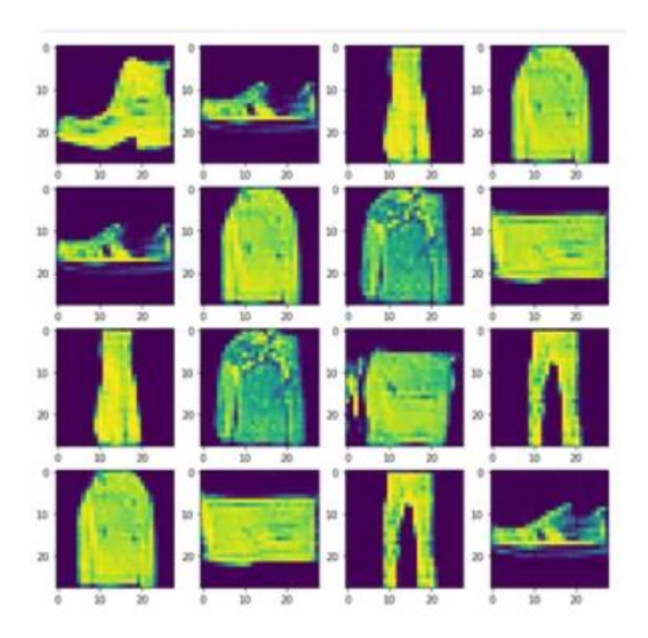
بعد تدريب GAN ومراجعة أدائه، يمكننا اختبار المولد عن طريق إنشاء صور أزياء جديدة وتصورها. أولا، سنقوم بتحميل أوزان المولد المدرب واستخدامه لتوليد صور جديدة.

```
# Load the weights of the trained generator
generator.load_weights('generator.h5')

# Generate new fashion images
imgs = generator.predict(tf.random.normal((16, 128, 1)))

# Plot the generated images
fig, ax = plt.subplots(ncols=4, nrows=4, figsize=(10, 10))
for r in range(4):
    for c in range(4):
        ax[r][c].imshow(imgs[(r + 1) * (c + 1) - 1])
```

- نقوم بتحميل أوزان المولد المدرب من الملف المحفوظ باستخدام generator.load_weights('generator.h5').
- نقوم بإنشاء صور أزياء جديدة عن طريق تمرير متجهات كامنة عشوائية إلى المولد. يفسر المولد هذه المتجهات العشوائية ويولد الصور المقابلة.
- نستخدم matplotlib لعرض الصور التي تم إنشاؤهافي شبكة 4 × 4.



حفظ النموذج

أخيرًا، إذا كنت راضيًا عن أداء GAN الخاص بك، فيمكنك حفظ نماذج المولد والمميز لاستخدامها في المستقبل.

```
# Save the generator and discriminator models
generator.save('generator.h5')
discriminator.save('discriminator.h5')
```

- نقوم بحفظ نماذج المولد والمميز على القرص باستخدام طريقة الحفظ.
- سيتم حفظ النماذجفي دليل العمل الحالي بأسماء الملفات "generator.h5" و"discriminator.h5" على التوالي.
- يتيح لك حفظ النماذج استخدامها لاحقًا لإنشاء المزيد من صور الأزياء أو لمواصلة عملية التدريب.

وبهذا تنتهي عملية بناء وتدريب شبكة GAN لتوليد صور الموضة باستخدام TensorFlow وKeras! تعد شبكات GAN نماذج قوية لتوليد بيانات واقعية ويمكن تطبيقها على مهام أخرى.

تذكر أن جودة الصور التي تم إنشاؤها تعتمد على معمارية GAN، وعدد فترات التدريب، وحجم مجموعة البيانات، والمعلمات الفائقة الأخرى. لا تترددفي تجربة GAN وضبطها لتحقيق نتائج أفضل. توليد سعيد Happy generating!

تحسينات إضافية والاتجاهات المستقبلية

تهانينا على إكمال GAN لإنشاء صور الموضة! الآن، دعنا نستكشف بعض التحسينات الإضافية والاتجاهات المستقبلية التي يمكنك وضعهافي الاعتبار لتحسين أداء GAN وإنشاء صور أزياء أكثر واقعية وتنوعًا.

ضبط المعلمات الفائقة

يمكن أن يؤثر ضبط المعلمات الفائقة Hyperparameter Tuning بشكل كبير على أداء شبكة GAN. قم بتجربة معدلات التعلم المختلفة، وأحجام الدفعات، وعدد فترات التدريب، وتكوينات المعمارية للمولد والمميز. يعد ضبط المعلمات الفائقة أمرًا ضروريًا لتدريب GAN، حيث يمكن أن يؤدي إلى تقارب أفضل ونتائج أكثر استقرارًا.

استخدام النمو التدريجي

تبدأ النمو التدريجي Progressive Growingفي تدريب GAN بصور منخفضة الدقة وتزيد دقة الصورة تدريجيًا أثناء التدريب. يساعد هذا الأسلوب على استقرار التدريب وينتج صورًا ذات جودة أعلى. يمكن أن يكون تنفيذ النمو التدريجي أكثر تعقيداً ولكنه يؤدي غالبًا إلى نتائج أفضل.

تنفيذ Wasserstein GAN (WGAN)

فكرفي استخدام Wasserstein GAN (WGAN) مع عقوبة التدرج gradient penalty بدلاً من خطأ GAN القياسية. يمكن أن توفر WGAN تدريبًا أكثر استقرارًا وتدرجات أفضل أثناء عملية التحسين. يمكن أن يؤدي هذا إلى تحسين التقارب وتقليل حالات انهيار الوضع.

زيادة البيانات

تطبيق تقنيات زيادة البيانات data augmentation على مجموعة بيانات التدريب. يمكن أن يشمل ذلك عمليات التدوير العشوائية والقلبات والترجمات والتحويلات الأخرى. تساعد زيادة البيانات شبكة GAN على التعميم بشكل أفضل ويمكن أن تمنع الضبط الزائد لمجموعة التدريب.

تضمين معلومات التسمية

إذا كانت مجموعة البيانات الخاصة بك تحتوي على معلومات التسمية label information (على سبيل المثال، فئات الملابس clothing categories)، فيمكنك محاولة تكييف GAN على معلومات التسمية أثناء التدريب. وهذا يعني تزويد المولد والمميز بمعلومات إضافية حول نوع الملابس، والتي يمكن أن تساعد GAN في إنشاء المزيد من صور الأزياء الخاصة بفئة معينة.

استخدام مميز مُدربة مسبقًا

يمكن أن يساعد استخدام المميز المدربة مسبقاً pretrained discriminator في تسريع التدريب وتحقيق استقرار GAN. يمكنك تدريب المميز على مهمة تصنيف باستخدام مجموعة بيانات MNIST للأزياء بشكل مستقل ثم استخدام المميز المدربة مسبقًا كنقطة بداية لتدريب GAN.

جمع مجموعة بيانات أكبر وأكثر تنوعًا

غالبًا ما يكون أداء شبكات GAN أفضل مع مجموعات البيانات الأكبر والأكثر تنوعًا. فكرفي جمع أو استخدام مجموعة بيانات أكبر تحتوي على مجموعة واسعة من أنماط الموضة والألوان والأنماط. يمكن لمجموعة البيانات الأكثر تنوعًا أن تؤدي إلى صور أكثر تنوعًا وواقعية.

اكتشاف معماريات مختلفة

تجربة مع مختلف معماريات المولد والمميز. هناك العديد من الأشكال المختلفة لشبكات GAN، مثل DCGAN (Deep Convolutional GAN)، وCGAN (Conditional GAN)، وStyleGAN. تتمتع كل معمارية بنقاط القوة والضعف الخاصة بها، ويمكن أن توفر تجربة نماذج مختلفة رؤى قيمة حول ما هو الأفضل لمهمتك المحددة.

استخدام نقل التعلم

إذا كان بإمكانك الوصول إلى نماذج GAN المدربة مسبقًا، فيمكنك استخدامها كنقطة بداية لـ GAN للأزياء الخاصة بك. يمكن أن يؤدي الضبط الدقيق Fine-tuning لشبكة GAN المدربة مسبقًا إلى توفير الوقت والموارد الحسابية مع تحقيق نتائج جيدة.

مراقبة انهيار الوضع

يحدث انهيار الوضع Mode collapse عندما ينهار المولد لإنتاج أنواع قليلة فقط من الصور. راقب العينات التي تم إنشاؤها بحثًا عن علامات انهيار الوضع واضبط عملية التدريب وفقًا لذلك إذا لاحظت هذا السلوك.

يعد بناء شبكات GAN وتدريبها عملية متكررة، وغالبًا ما يتطلب تحقيق نتائج مبهرة التجريب والضبط الدقيق. استمرفي الاستكشاف والتعلم وتكييف شبكة GAN الخاصة بك لإنشاء صور أزياء أفضل!

بهذا نختتم رحلتنافي إنشاء صورة أزياء GAN باستخدام TensorFlow وKeras. لا تترددفي استكشاف تطبيقات GAN الأخرى، مثل إنشاء أعمال فنية أو وجوه أو كائنات ثلاثية الأبعاد. أحدثت شبكات GAN ثورةفي مجال النمذجة التوليدية ولا تزال مجالًا مثيرًا للبحث والتطويرفي مجتمع الذكاء الاصطناعي. حظا سعيدافي مشاريع GAN المستقبلية الخاصة بك!

الاستنتاج

في الختام، تمثل شبكات الخصومة التوليدية (GANs) تقنية متطورةفي الذكاء الاصطناعي أحدثت ثورة في إنشاء عينات البيانات الاصطناعية. خلال هذا الدليل، اكتسبنا فهمًا عميقًا لشبكات GAN ونجحنا في بناء مشروع رائع: **شبكة GAN لإنشاء صور الموضة**.

1. **شبكات GAN:** تتكون شبكات GAN من شبكتين عصبيتين، شبكة المولد والمميز، والتي تستخدم التدريب التنافسي لإنشاء عينات بيانات واقعية.
2. **هدف المشروع:** نحن نهدف إلى تطوير GAN التي تولد صور أزياء تشبه تلك الموجودة في مجموعة بيانات Fashion MNIST.
3. **مجموعة البيانات:** كانت مجموعة بيانات Fashion MNIST، التي تحتوي على صور ذات تدرج رمادي لعناصر الموضة، بمثابة الأساس لمولد صور الأزياء الخاص بنا.
4. **بناء شبكة GAN:** قمنا ببناء المولد والمميز باستخدام واجهة برمجة التطبيقات التسلسلية Sequential API الخاصة بـ TensorFlow، والتي تتضمن طبقات مثل Dense وConv2D وLeakyReLU.
5. **حلقة تدريب GAN:** استخدمنا حلقة تدريب مصممة بعناية لتحسين المولد والمميز بشكل متكرر.
6. **التحسينات:** استكشفنا العديد من التقنيات لتحسين أداء GAN، بمافي ذلك ضبط المعلمات الفائقة، والنمو التدريجي، وWasserstein GAN، وزيادة البيانات، وGAN المشروطة.

7. **التقييم:** ناقشنا مقاييس التقييم مثل Inception Score وFID لتقييم جودة صور الأزياء التي تم إنشاؤها بشكل موضوعي.
8. **الضبط الدقيق ونقل التعلم:** من خلال الضبط الدقيق للمولد واستخدام النماذج المدربة مسبقًا، كنا نهدف إلى تحقيق توليد صور أزياء أكثر تنوعًا وواقعية.
9. **الاتجاهات المستقبلية:** هناك فرص لا حصر لها لمزيد من التحسينات والبحث في شبكات GAN، بما في ذلك تحسين المعلمات الفائقة، والنمو التدريجي، وWasserstein GAN، والمزيد.

باختصار، قدم هذا الدليل الشامل أساسًا متينًا لفهم شبكات GAN، وتعقيدات تدريبها، وكيف يمكن تطبيقها على توليد صور الموضة. لقد أظهرنا إمكانية إنشاء بيانات اصطناعية متطورة وواقعية من خلال استكشاف التقنيات والتطورات المختلفة. ومع تطور شبكات GAN، فإنها على استعداد لإحداث تحول في مختلف الصناعات، بما في ذلك الفن والتصميم والرعاية الصحية والمزيد. إن احتضان القوة المبتكرة لشبكات GAN واستكشاف إمكانياتها اللامحدودة هو مسعى مثير سيشكل بلا شك مستقبل الذكاء الاصطناعي.

المصدر:

https://www.analyticsvidhya.com/blog/2023/07/gans-in-vogue-a-step-by-step-guide-to-fashion-image-generation/

7) توليد الصور باستخدام شبكات الخصومة التوليدية Images Generation using GANs

المقدمة

في هذه المقالة، نستكشف تطبيق شبكات GAN في TensorFlow لإنشاء إصدارات فريدة من الأرقام المكتوبة بخط اليد. يتكون إطار عمل GAN من مكونين رئيسيين: المولد generator والمميز discriminator. يقوم المولد بإنشاء صور جديدة بطريقة عشوائية، بينما تم تصميم المميز للتمييز بين الصور الأصلية والمقلدة. من خلال تدريب GAN، حصلنا على مجموعة من الصور التي تشبه إلى حد كبير الأرقام المكتوبة بخط اليد. الهدف الأساسي من هذه المقالة هو تحديد الإجراء الخاص ببناء وتقييم شبكات GAN باستخدام مجموعة بيانات MNIST.

أهداف التعلم

1) توفر هذه المقالة مقدمة شاملة لشبكات الخصومة التوليدية Generative Adversarial Networks (GANs) وتستكشف تطبيقاتهافي توليد الصور.
2) الهدف الرئيسي من هذا البرنامج التعليمي هو توجيه القراء خلال عملية خطوة بخطوة لإنشاء شبكة GAN باستخدام مكتبة TensorFlow. ويغطي تدريب GAN على مجموعة بيانات MNIST لإنشاء صور جديدة للأرقام المكتوبة بخط اليد.
3) يناقش المقال معمارية ومكونات شبكات GAN، بمافي ذلك المولدات والمميزات، لتعزيز فهم القراء لأعمالهم الأساسية.
4) للمساعدةفي التعلم، تتضمن المقالة أمثلة التعليمات البرمجية التي توضح المهام المختلفة، مثل قراءة مجموعة بيانات MNIST ومعالجتها مسبقًا، وبناء بنية GAN، وحساب دوال الخطأ، وتدريب الشبكة، وتقييم النتائج.
5) علاوة على ذلك، يستكشف المقال النتيجة المتوقعة لشبكات GAN، وهي عبارة عن مجموعة من الصور التي تحمل تشابهًا مذهلاً مع الأرقام المكتوبة بخط اليد.

ماذا نبني؟

يعد إنشاء صور جديدة باستخدام قواعد بيانات الصور الموجودة مسبقًا سمة بارزة للنماذج المتخصصة التي تسمى شبكات الخصومة التوليدية (GANs). تتفوق شبكات GAN في إنتاج صور غير خاضعة للإشراف أو شبه خاضعة للإشراف من خلال الاستفادة من مجموعات بيانات الصور المتنوعة.

تستغل هذه المقالة إمكانات توليد الصور لشبكات GAN لإنشاء أرقام مكتوبة بخط اليد. تتضمن المنهجية تدريب الشبكة على قاعدة بيانات رقمية مكتوبة بخط اليد.في هذه المقالة التعليمية، سنقوم

ببناء شبكة GAN بدائية باستخدام مكتبة Tensorflow، وإجراء التدريب على مجموعة بيانات MNIST، وإنشاء صور جديدة للأرقام المكتوبة بخط اليد.

كيف نقوم بإعداد هذا؟

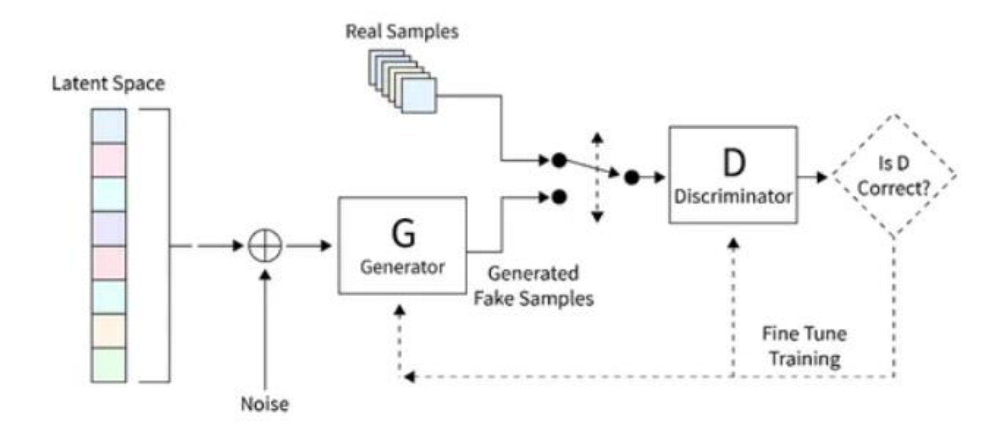
يتمحور التركيز الأساسي لهذه المقالة حول تسخير إمكانات توليد الصور لشبكات GAN. يبدأ الإجراء بالتحميل والمعالجة المسبقة لقاعدة بيانات الصور لتسهيل عملية تدريب GAN. بمجرد تحميل البيانات بنجاح، نبدأفي إنشاء نموذج GAN وتطوير الكود اللازم للتدريب والاختبار.في القسم التالي، يتم توفير تعليمات مفصلة حول تنفيذ هذه الدالة وإنشاء صورة جديدة باستخدام قاعدة بيانات MNIST.

بناء النموذج

يتكون نموذج GAN الذي نهدف إلى بناءه من عنصرين مهمين:

- المولد Generator: هذا المكون مسؤول عن إنشاء صور جديدة.
- المميز Discriminator: يقوم هذا المكون بتقييم جودة الصورة التي تم إنشاؤها.

تظهر المعمارية العامة التي سنقوم بتطويرها لإنشاء الصور باستخدام GANفي الرسم البياني أدناه. يقدم القسم التالي وصفًا موجزًا لكيفية قراءة قاعدة البيانات وإنشاء المعمارية المطلوبة وحساب دالة الخطأ وتدريب الشبكة. بالإضافة إلى ذلك، يتم توفير التعليمات البرمجية لفحص الشبكة وإنشاء صور جديدة.



قراءة مجموعة البيانات

تتمتع مجموعة بيانات MNIST بأهمية كبيرة في مجال الرؤية الحاسوبية computer vision وتضم مجموعة كبيرة من الأرقام المكتوبة بخط اليد بأبعاد 28 × 28 بكسل. أثبتت مجموعة البيانات هذه أنها مثالية لتنفيذ GAN نظرًا لتنسيق الصورة أحادي القناة ذي التدرج الرمادي.

يوضح مقتطف التعليمات البرمجية اللاحق استخدام دالة مضمنة في Tensorflow لتحميل مجموعة بيانات MNIST. عند التحميل الناجح، نبدأ في تسوية normalize الصور وإعادة تشكيلها إلى تنسيق ثلاثي الأبعاد. يتيح هذا التحويل المعالجة الفعالة لبيانات الصور ثنائية الأبعاد ضمن معمارية GAN. بالإضافة إلى ذلك، يتم تخصيص الذاكرة لكل من بيانات التدريب والتحقق من الصحة.

يتم تعريف شكل كل صورة على أنه مصفوفة 28x28x1، حيث يمثل البعد الأخير عدد القنوات في الصورة. نظرًا لأن مجموعة بيانات MNIST تشتمل على صور ذات تدرج رمادي، فلدينا قناة واحدة فقط.

في هذه الحالة تحديداً، قمنا بتعيين حجم المساحة الكامنة latent space، المشار إليها بـ "zsize"، على 100. ويمكن تعديل هذه القيمة وفقًا لمتطلبات أو تفضيلات محددة.

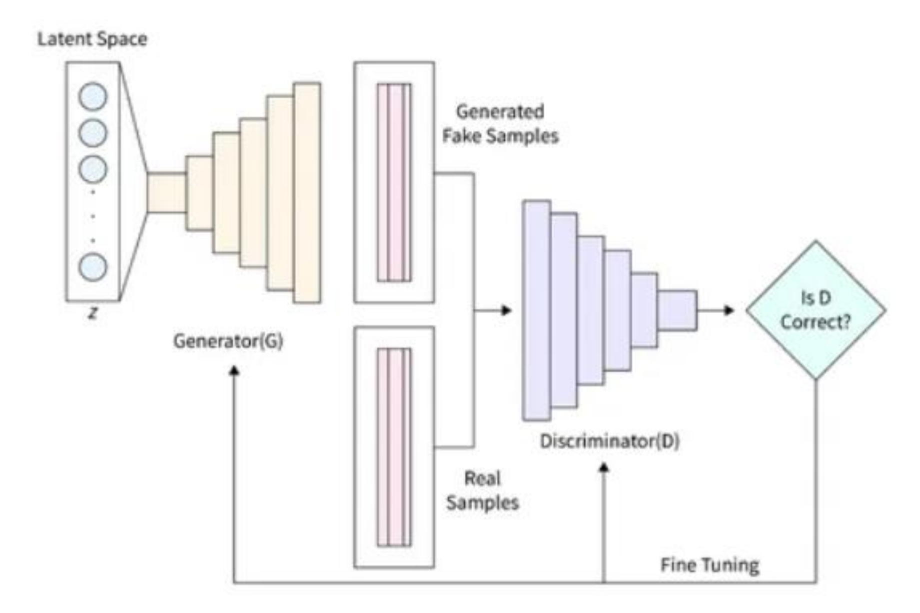
```
from __future__ import print_function, division
from keras.datasets import mnist
from keras.layers import Input, Dense, Reshape, Flatten,
Dropout
from keras.layers import BatchNormalization, Activation,
ZeroPadding2D
from keras.layers import LeakyReLU
from keras.layers.convolutional import UpSampling2D, Conv2D
from keras.models import Sequential, Model
from keras.optimizers import Adam, SGD
import matplotlib.pyplot as plt
import sys
import numpy as np

num_rows = 28
num_cols = 28
num_channels = 1
input_shape = (num_rows, num_cols, num_channels)
z_size = 100

(train_ims, _), (_, _) = mnist.load_data()
train_ims = train_ims / 127.5 - 1.
train_ims = np.expand_dims(train_ims, axis=3)

valid = np.ones((batch_size, 1))
                                            fake = np.zeros((batch_size, 1))
```

تعريف المولد



يلعب المولد (G) دورًا حاسمًافي شبكات GAN لأنه مسؤول عن إنشاء صور واقعية يمكن أن تخدع المُميِّز. إنه بمثابة المكون الأساسي لتكوين الصورةفي شبكات GAN.في هذه الدراسة، نستخدم معمارية محددة للمولد، والتي تتضمن طبقة متصلة بالكامل fully connected (FC) وتستخدم تنشيط Leaky ReLU. ومع ذلك، تجدر الإشارة إلى أن الطبقة الأخيرة من المولد تستخدم تنشيط TanH بدلاً من LeakyReLU. تم إجراء هذا التعديل للتأكد من أن الصورة التي تم إنشاؤها موجودة ضمن نفس الفاصل الزمني (ـ1، 1) مثل قاعدة بيانات MNIST الأصلية.

```
def build_generator():
    gen_model = Sequential()
    gen_model.add(Dense(256, input_dim=z_size))
    gen_model.add(LeakyReLU(alpha=0.2))
    gen_model.add(BatchNormalization(momentum=0.8))
    gen_model.add(Dense(512))
    gen_model.add(LeakyReLU(alpha=0.2))
    gen_model.add(BatchNormalization(momentum=0.8))
    gen_model.add(Dense(1024))
    gen_model.add(LeakyReLU(alpha=0.2))
    gen_model.add(BatchNormalization(momentum=0.8))
    gen_model.add(Dense(np.prod(input_shape),
activation='tanh'))
    gen_model.add(Reshape(input_shape))

    gen_noise = Input(shape=(z_size,))
    gen_img = gen_model(gen_noise)
```

```
        return Model(gen_noise, gen_img)
```

تعريف المميز

في شبكة الخصومة التوليدية (GAN)، يؤدي المُميِّز (D) المهمة الحاسمة المتمثلةفي التمييز بين الصور الحقيقية والصور المولدة من خلال تقييم صحتها واحتماليتها. يمكن اعتبار هذا المكون بمثابة مشكلة تصنيف ثنائية binary classification problem. لمعالجة هذه المهمة، يمكننا استخدام معمارية شبكة مبسطة تشتمل على طبقات متصلة بالكامل (FC)، وتنشيط Leaky ReLU، وطبقات Dropout. من المهم الإشارة إلى أن الطبقة الأخيرة من Discriminator تتضمن طبقة FC متبوعة بتنشيط Sigmoid. تنتج دالة التنشيط Sigmoid احتمالية التصنيف المطلوبة.

```
def build_discriminator():
    disc_model = Sequential()
    disc_model.add(Flatten(input_shape=input_shape))
    disc_model.add(Dense(512))
    disc_model.add(LeakyReLU(alpha=0.2))
    disc_model.add(Dense(256))
    disc_model.add(LeakyReLU(alpha=0.2))
    disc_model.add(Dense(1, activation='sigmoid'))

    disc_img = Input(shape=input_shape)
    validity = disc_model(disc_img)
                            return Model(disc_img, validity)
```

حساب دالة الخطأ

من أجل ضمان عملية جيدة لتوليد الصورفي شبكات GAN، من المهم تحديد المقاييس المناسبة لتقييم أدائها. حدد هذه المعلمة بواسطة دالة الخطأ loss function.

والمميز مسؤول عن تقسيم الصورة المولدة إلى حقيقية أو مزيفة وإعطاء احتمال أن تكون حقيقية. لتحقيق هذا الاختلاف، يهدف المميز إلى تعظيم الدالة D(x) عند تقديمها بصورة حقيقية وتقليل D(G(z)) عند تقديمها بصورة مزيفة.

ومن ناحية أخرى، فإن الغرض من المولد هو خداع المُميِّز من خلال إنشاء صورة واقعية يمكن إساءة تفسيرها. رياضيًا، يتضمن ذلك قياس D(G(z)). ومع ذلك، فإن الاعتماد فقط على هذا المكون كدالة خطأ يمكن أن يتسببفي زيادة ثقة الشبكة بالنتائج الخاطئة. لحل هذه المشكلة، نستخدم سجل دالة الخطأ (D(G(z))).

يمكن التعبير عن دالة التكلفة الإجمالية لـ GAN لإنشاء صورة باعتبارها لعبة بسيطة:

```
min_G max_D V(D,G) = E(xp_data(x))(log(D(x))] + E(zp(z))(log(1 - D(G(z)))])
```

يتطلب تدريب GAN هذا توازنًا جيدًا ويمكن أن يكون بمثابة مباراة بين خصمين. يسعى كل طرف للتأثير على الآخر والتفوق عليه من خلال ممارسة لعبة MinMax.

يمكننا استخدام خطأ الانتروبيا المتقاطعة الثنائية Binary Cross Entropy Loss لتنفيذ المولد والمميز.

لتنفيذ المولد والمميز، يمكننا الاستفادة من خطأ الانتروبيا المتقاطعة الثنائية.

```
# discriminator
disc= build_discriminator()
disc.compile(loss='binary_crossentropy',
    optimizer='sgd',
    metrics=['accuracy'])

z = Input(shape=(z_size,))

# generator
img = generator(z)

disc.trainable = False

validity = disc(img)

# combined model
combined = Model(z, validity)
                combined.compile(loss='binary_crossentropy',
                                           optimizer='sgd')
```

تحسين الخطأ

لتسهيل تدريب الشبكة، هدفنا هو إشراك GANفي لعبة MinMax. تدور عملية التعلم هذه حول تحسين أوزان الشبكة من خلال استخدام التدرج الاشتقاقي Gradient Descent. من أجل تسريع عملية التعلم ومنع التقارب convergence مع suboptimal loss landscapes، يتم استخدام التدرج الاشتقاقي العشوائي (SGD) Stochastic Gradient Descent.

نظرًا لأن المميز والمولد لهما اخطاء مختلفة، لا يمكن لدالة خطأ واحدة تحسين كلا النظامينفي وقت واحد. وبالتالي، استخدم دوال الخطأ المنفصلة لكل نظام.

```
def intialize_model():
    disc= build_discriminator()
    disc.compile(loss='binary_crossentropy',
        optimizer='sgd',
        metrics=['accuracy'])

    generator = build_generator()

    z = Input(shape=(z_size,))
    img = generator(z)

    disc.trainable = False
```

```
    validity = disc(img)

    combined = Model(z, validity)
    combined.compile(loss='binary_crossentropy',
optimizer='sgd')
    return disc, Generator, and combined
```

بعد تحديد جميع الميزات المطلوبة، يمكننا تدريب النظام وتحسين الخطأ. فيما يلي خطوات تدريب GAN لإنشاء صورة:

- قم بتحميل الصورة وقم بإنشاء صوت عشوائي بنفس حجم الصورة المحملة.
- التمييز بين الصورة التي تم تحميلها والصوت الناتج والنظر في إمكانية كونها حقيقية أو مزيفة.
- قم بإنتاج ضوضاء عشوائية أخرى بنفس الحجم وقم بتوفيرها كمدخل للمولد.
- تدريب المولد لمدة محددة.
- كرر هذه الخطوات حتى تصبح الصورة مرضية.

```
def train(epochs, batch_size=128, sample_interval=50):
    # load images
    (train_ims, _), (_, _) = mnist.load_data()
    # preprocess
    train_ims = train_ims / 127.5 - 1.
    train_ims = np.expand_dims(train_ims, axis=3)

    valid = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))
    # training loop
    for epoch in range(epochs):

        batch_index = np.random.randint(0,
train_ims.shape[0], batch_size)
        imgs = train_ims[batch_index]
    # create noise
        noise = np.random.normal(0, 1, (batch_size, z_size))
    # predict using a Generator
        gen_imgs = gen.predict(noise)
    # calculate loss functions
        real_disc_loss = disc.train_on_batch(imgs, valid)
        fake_disc_loss = disc.train_on_batch(gen_imgs, fake)
        disc_loss_total = 0.5 * np.add(real_disc_loss,
fake_disc_loss)

        noise = np.random.normal(0, 1, (batch_size, z_size))

        g_loss = full_model.train_on_batch(noise, valid)

    # save outputs every few epochs
```

```
        if epoch % sample_interval == 0:
            one_batch(epoch)
```

توليد أرقام مكتوبة بخط اليد

باستخدام مجموعة بيانات MNIST، يمكننا إنشاء دالة مساعدة لإنشاء تنبؤات لمجموعة من الصور باستخدام المولد. تقوم هذه الدالة بتوليد صوت عشوائي، وتزويده للمولد، وتشغيله لعرض الصورة التي تم إنشاؤها وحفظهافي مجلد خاص. يوصى بتشغيل هذه الدالة المساعدة بشكل دوري، مثل كل 200 دورة، لمراقبة تقدم الشبكة. التنفيذ أدناه:

```
def one_batch(epoch):
    r, c = 5, 5
    noise_model = np.random.normal(0, 1, (r * c, z_size))
    gen_images = gen.predict(noise_model)

    # Rescale images 0 - 1
    gen_images = gen_images*(0.5) + 0.5

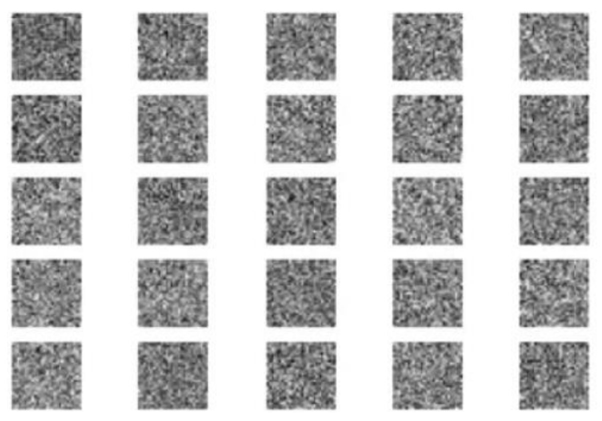
    fig, axs = plt.subplots(r, c)
    cnt = 0
    for i in range(r):
        for j in range(c):
            axs[i,j].imshow(gen_images[cnt, :,:,0],
cmap='gray')
            axs[i,j].axis('off')
            cnt += 1
    fig.savefig("images/%d.png" % epoch)
    plt.close()
```

في تجربتنا، قمنا بتدريب GAN لما يقرب من 10000 حقبة باستخدام حجم دفعة يبلغ 32. لتتبع تقدم التدريب، قمنا بحفظ الصور التي تم إنشاؤها كل 200 فترة وقمنا بتخزينهافي مجلد مخصص يسمى "images".

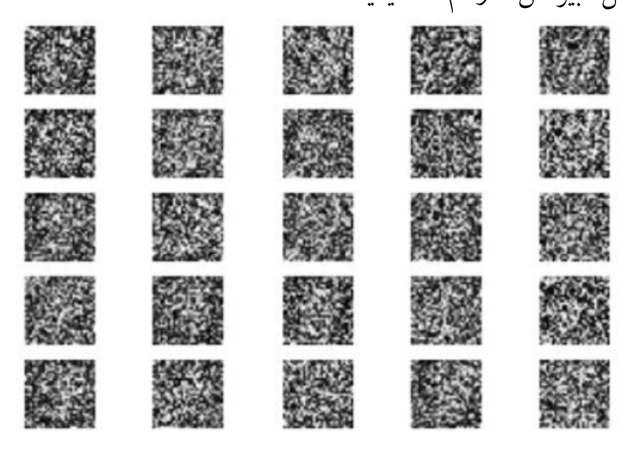
```
disc, gen, full_model = intialize_model()
train(epochs=10000, batch_size=32, sample_interval=200)
```

الآن، دعونا نتفحص نتائج محاكاة GANفي مراحل مختلفة: التهيئة initialization، 400 فترة، 5000 فترة، والنتيجة النهائية عند 10000 فترة.

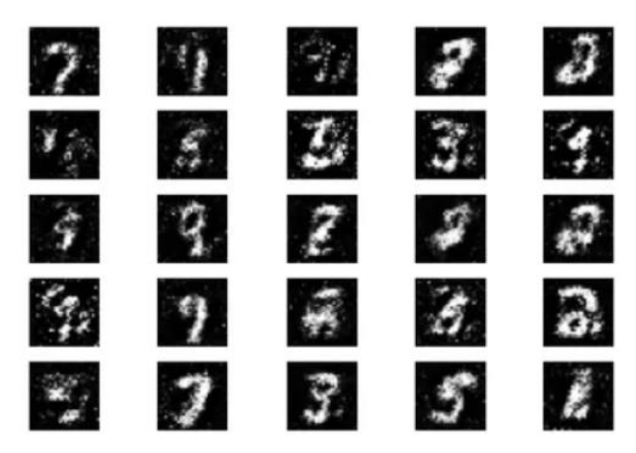
في البداية، نبدأ بالضوضاء العشوائية random noise كمدخل للمولد.



وبعد 400 فترة من التدريب، يمكننا ملاحظة بعض التقدم، على الرغم من أن الصور التي تم إنشاؤها لا تزال تختلف بشكل كبير عن الأرقام الحقيقية.



بعد التدريب لمدة 5000 فترة، يمكننا أن نلاحظ أن الأرقام الناتجة بدأت تشبه مجموعة بيانات MNIST.



أكمل 10.000 فترة تدريب كاملة، نحصل على المخرجات التالية.



تشبه هذه الصور التي تم إنشاؤها إلى حد كبير بيانات الأرقام المكتوبة بخط اليد لتدريب الشبكة. من المهم ملاحظة أن هذه الصور ليست جزءًا من مجموعة التدريب وتم إنشاؤها بالكامل بواسطة الشبكة.

الخطوات التالية

الآن بعد أن حققنا نتائج جيدةفي إنشاء صور GAN، هناك العديد من الطرق التي يمكننا من خلالها تحسينها.في نطاق هذه المناقشة، قد نفكرفي تجربة معلمات مختلفة. وفيما يلي بعض الاقتراحات:

- استكشف قيمًا مختلفة لمتغير المساحة الكامن z_size لمعرفة ما إذا كان يزيد من الكفاءة.
- زيادة عدد فترات التدريب إلى أكثر من 10000. إن مضاعفة مدة التدريب أو مضاعفتها ثلاث مرات قد تكشف عن نتائج محسنة أو متدهورة.
- حاول استخدام مجموعات بيانات مختلفة مثل MNIST للأزياء أو MNIST المتحرك. نظرًا لأن مجموعات البيانات هذه لها نفس معمارية MNIST، فقم بتعديل التعليمات البرمجية الموجودة لدينا.
- فكرفي تجربة معماريات بديلة مثل CycleGun وDCGAN وغيرها. قد يكون تعديل دوال المولد والمميز كافياً لاستكشاف هذه النماذج.

ومن خلال تنفيذ هذه التغييرات، يمكننا تعزيز قدرات شبكات GAN واستكشاف إمكانيات جديدةفي توليد الصور.

تشبه هذه الصور التي تم إنشاؤها إلى حد كبير بيانات الأرقام المكتوبة بخط اليد والتي يتم استخدامها لتدريب الشبكة. هذه الصور ليست جزءًا من مجموعة التدريب وتم إنشاؤها بالكامل بواسطة الشبكة.

الاستنتاج

باختصار، يُعد GAN نموذجًا قويًا للتعلم الآلي قادرًا على إنشاء صور جديدة بناءً على قواعد البيانات الموجودة.في هذا البرنامج التعليمي، أظهرنا كيفية تصميم وتدريب شبكة GAN بسيطة باستخدام مكتبة Tensorflow كمثال وقاعدة بيانات MNIST.

- يتكون GAN من عنصرين مهمين: المولد، وهو المسؤول عن توليد صور جديدة من المدخلات العشوائية، والمميز، الذي يهدف إلى التمييز بين الصور الحقيقية والمزيفة.
- نجحنا من خلال عملية التعلمفي إنشاء مجموعة من الصور التي تشبه إلى حد كبير الأرقام المكتوبة بخط اليد، كما هو موضحفي الصورة النموذجية.
- لتحسين أداء GAN، نقدم مقاييس مطابقة ودوال الخطأ التي تساعدفي التمييز بين الصور الحقيقية والمزيفة. من خلال تقييم شبكات GAN على البيانات غير المرئية واستخدام المولدات، يمكننا إنشاء صور جديدة لم يتم رؤيتها من قبل.
- بشكل عام، توفر شبكات GAN إمكانيات مثيرة للاهتمامفي توليد الصور ولديها إمكانات كبيرة للعديد من التطبيقات مثل التعلم الآلي والرؤية الحاسوبية.

المصدر:

https://www.analyticsvidhya.com/blog/2023/06/using-gans-in-tensorflow-generate-images/

8) توليد وجه الأنمي باستخدام شبكات الخصومة التوليدية
Anime Face Generation using Generative Adversarial Networks

في السنوات الأخيرة، أصبحت شبكات الخصومة التوليدية Generative adversarial networks (GAN) مجالًا بحثيًا شائعًا للغاية لتوليد بيانات اصطناعية artificial data، خاصة لتوليد الصور.

كثير من الناس يحبون الرسوم المتحركة الأنمي anime characters ويريدون أن يكون لديهم شخصيات أنمي مخصصة خاصة بهم. لكن إنشاء شخصيات الأنمي يحتاج إلى مهارات رسم وتصميم احترافية لا يستطيع الجميع القيام بها. من المهم أن يكون لديك طريقة لإنشاء وجوه أنمي بدون مهارات فنية.

سأشرح اليوم كيفية بناء نموذج GAN لإنشاء صور وجوه لشخصيات الأنمي.

ما هو GAN؟

تعد شبكات الخصومة التوليدية (GANs) من تقنيات التعلم الآلي الشائعة والقوية المستخدمةفي إنشاء الصور والنصوص والفيديو والصوت. منذ تقديم أول شبكة GANفي عام 2014، تطور مجال البحث هذا بسرعة، ويظل أكثر معمارية الشبكات العصبية المستخدمة اليوم مرونة.

يتم تصنيف GAN ضمن التعلم غير الخاضع للأشراف. وهذا هو أحد الأسباب الرئيسية التي دفعت الكثير من الباحثين إلى تنفيذ عملهم باستخدام النماذج التوليدية generative models.

تستخدم GAN شبكتين عصبيتين مستقلتين لتوليد بيانات جديدة، تسمى المولد Generator والمميز Discriminator.

ماذا تفعل الشبكة العصبية للمولدات؟

تقوم الشبكة العصبية للمولدات بإنشاء البيانات الاصطناعية (عينات مزيفة fake samples) باستخدام ضوضاء عشوائية random noise.

ماذا تفعل الشبكة العصبية للمميزات؟

تعمل الشبكة العصبية للمميزات كمصنف ثنائي وتصنف عينة الإدخال على أنها حقيقية real أو مزيفة fake.

مجموعة البيانات



من المهم الحصول على بيانات جيدة للحصول على نتائج جيدة.

في هذا المشروع، أستخدم مجموعة بيانات الصور العامة الخاصة بـ Kaggle، Anime Face Dataset NTU-MLDS.

تتكون مجموعة البيانات من 36740 صورة عالية الجودة لوجوه الأنمي. جميع الصور هي صور ملونة بحجم 64 × 64 بكسل.

قبل البدء، لنقم بتنزيل مجموعة البيانات باستخدام Kaggle API.

```
def download_data():
    kaggle.api.authenticate()
```

```
    kaggle.api.dataset_download_files('lunarwhite/anime-
face-dataset-ntumlds', path='./', unzip=True)
download_data()
```

إذا لم تكن على دراية بتنزيل بيانات Kaggle، فاتبع هذه الخطوات.

إضافة الصور إلى القائمة وتصور الصور.

```
from PIL import Image
import random
import glob
image_list = []
rows = []
for filename in glob.glob('./images/*.jpg'):
    im=Image.open(filename)
    rows.append([filename])
    image_list.append(filename)
print(len(image_list))
def gallery(array, ncols=8):
    nindex, height, width, intensity = array.shape
    nrows = nindex//ncols
    assert nindex == nrows*ncols
    result = (array.reshape(nrows, ncols, height, width,
intensity)
              .swapaxes(1,2)
              .reshape(height*nrows, width*ncols,
intensity))
    return result
def make_array():
    arr = []
    # Randomly select 64 images to visualize
    for i in range(64):
        random_image = random.choice(image_list)
        arr.append(np.asarray(
                    Image.open(random_image).convert('RGB')
                    ))
    return np.array(arr)

array = make_array()
result = gallery(array)
plt.figure(figsize=(8,8))
plt.imshow(result)
plt.show()
```

المعالجة المسبقة وتحميل البيانات

تعد المعالجة المسبقة للبيانات Data preprocessing إحدى المهام الرئيسية للتأكد من أن البيانات مناسبة لتدريب النموذج.



أولاً، قص كل صورة في المركز وقم بتغيير حجم الصورة إلى 64 مع الاستيفاء الثنائي الخطي، ثم حول الصورة بنطاق بكسل [0, 255] إلى كائن tensor وأخيرًا قم بتحويل الصور إلى قيم متوسط الصورة هو 0.5 والانحراف المعياري للصورة هو 0.5. سيؤدي هذا إلى تسوية الصورة في النطاق [1, 1−].

```
df = pd.DataFrame(rows)
df.to_csv('data.csv', index=False, header = None)

# Preprocessing
#The batch_size is defined based on the memory of GPU or
CPU.
batch_size = 128
stats = (0.5, 0.5, 0.5), (0.5, 0.5, 0.5)
transform = transforms.Compose([transforms.CenterCrop(64),
                                transforms.Resize(64,
interpolation=2),
                                transforms.ToTensor(),

transforms.Normalize(*stats)])
def denorm(img_tensors):
    return img_tensors * stats[1][0] + stats[0][0]

# loading data .
class AnimeData(Dataset):
    """
    Wrap the data into a Dataset class, and then pass it to
the DataLoader
    :__init__: Initialization data
    :__getitem__: support the indexing such that dataset[i]
can be used to get ith sample
    :__len__: return the size of the dataset.
    """
    def __init__(self, root, transform=None):
        self.frame = pd.read_csv(root, header=None)
        self.transform = transform

    def __len__(self):
        return len(self.frame)

    def __getitem__(self, index):
```

```
        image_name = self.frame.iloc[index, 0]
        image = Image.open(image_name)
        image = self.transform(image)
        return image

trainset = AnimeData(root='./data.csv', transform=transform)
trainloader = DataLoader(trainset, batch_size, shuffle=True,
num_workers=0)
```

حدد مجموعة البيانات (AnimeData) وقم بتحميل صور التدريب باستخدام DataLoader.

التحقق من توفر GPU ونقل البيانات

نحن هنا نتحقق مما إذا كانت وحدة معالجة الرسوميات (GPU) للكمبيوتر متاحة ونقل البيانات إلى GPU (أو وحدة المعالجة المركزية CPU).

استخدم GPU لتسريع البرنامج.

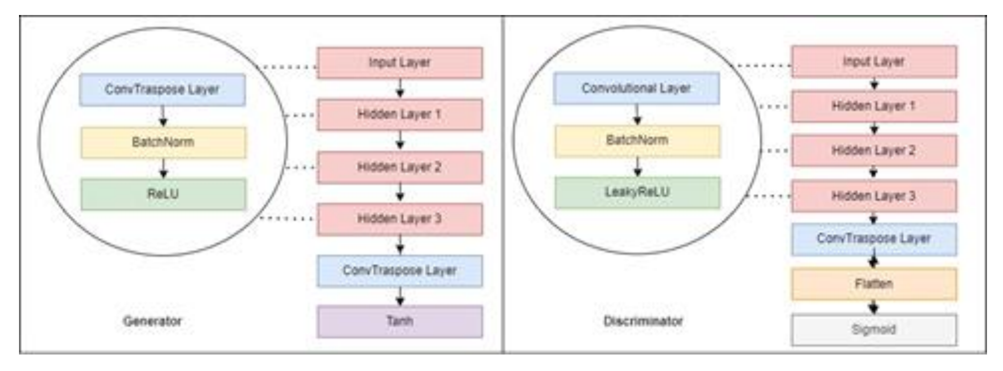
```
if torch.cuda.is_available():
    device=torch.device('cuda')
else:
    device=torch.device('cpu')
```

تعريف GAN

وهنا الجزء المهم !!!

معمارية الشبكات العصبية

كما ذكرنا أعلاه، تحتوي معمارية GAN على شبكتين عصبيتين تلافيفيتين عميقتين لتوليد الصور، ومولد، ومميز. وباستخدام الضوضاء كمدخل، يقوم نموذج المولد بإنشاء صور "مزيفة fake".



في هذا المشروع، أستخدم مولدًا يحتوي على 5 طبقات تلافيفية منقولة transpose convolutional layers، وسوف يقوم بأخذ عينات من المدخلات ويتعلم كيفية ملء تفاصيل الصورة أثناء عملية التدريب.

تم استخدام تسوية الدُفعات Batch normalization في نموذج المولد لتوحيد standardize المدخلات في الطبقات.

تم استخدام دالة تنشيط الوحدة الخطية المصححة Rectified linear unit (ReLU) في كل طبقة مخفية ودالة تنشيط Tanh في طبقة الإخراج.

إخراج المولد هو الصورة الجديدة بنفس الحجم كما في عينات بيانات التدريب، 3 x 64 x 64.

```
# Create your Generator model
latent_size = 128
class Generator(nn.Module):
    def __init__(self,latent_size):
        super(Generator,self).__init__()
        """
        Initialize the Generator Module
        :param latent_size: The length of the input latent
vector
        """
        self.conv_block1 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=latent_size,
out_channels=512, kernel_size=4, stride=1, padding=0),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.ConvTranspose2d(in_channels=512,
out_channels=256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.ConvTranspose2d(in_channels=256,
out_channels=128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.ConvTranspose2d(in_channels=128,
out_channels=64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(in_channels=64,
out_channels=3, kernel_size=4, stride=2, padding=1),
            nn.Tanh()
        )

    def forward(self,x):
        """
        Forward propagation of the neural network
        :param x: The input to the neural network
        :return: A 3x64x64 Tensor image as output
        """
        x = self.conv_block1(x)
        return x
```

```
G=Generator(latent_size).to(device)
# random latent tensors
noise = torch.randn(batch_size, latent_size, 1, 1)
# use generator model to generate fake image
fake_images = G(noise)

#visualize the fake images by function show_images

def show_images(images, nmax=64):
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(denorm(images.detach()[:nmax]),
nrow=8).permute(1, 2, 0))

show_images(fake_images)

for real_images in tqdm(trainloader):
    real_images=(real_images).to(device)

show_images(real_images)
```

يحتوي المُميِّز أيضًا على 5 طبقات تلافيفية. على غرار المولد، تم استخدام تسوية الدُفعات للطبقات المخفية ودالة تنشيط LeakyReLU ودالة التنشيط sigmoid.

مدخلات المميز هي 64 x 64 x 3 صور موتر، ويعطي المميز قيمة واحدة كمخرجات تشير إلى ما إذا كانت الصورة المعطاة مزيفة أم لا.

```
# Create your Discriminator model
class Discriminator(nn.Module):
    def __init__(self,inchannels):
        super(Discriminator,self).__init__()
        """
        Initialize the Discriminator Module
        :param inchannels: The depth of the first
convolutional layer
        """
        self.conv_block1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64,
kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, inplace = True),
            nn.Conv2d(in_channels=64, out_channels=128,
kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace = True),
            nn.Conv2d(in_channels=128, out_channels=256,
kernel_size=4, stride=2, padding=1),
```

```
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace = True),
            nn.Conv2d(in_channels=256, out_channels=512,
kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace = True),
            nn.Conv2d(in_channels=512, out_channels=1,
kernel_size=4, stride=2, padding=0),
            nn.Flatten(),
            nn.Sigmoid()
        )

    def forward(self,x):
        """
        Forward propagation of the neural network
        :param x: The input to the neural network
        :return: Discriminator logits; the output of the
neural network
        """
        x = self.conv_block1(x)
        return x

D=Discriminator(3).to(device)
```

دالة الخطأ وخوارزمية التحسين والمعلمات الفائقة

يتم تدريب شبكات المولدات والمميزاتفي وقت واحد. يتم حساب خطأ كلتا الشبكتين العصبيتين باستخدام دالة خطأ الخطأ التربيعي المتوسط mean squared error (MSE).

المُحسِّنات Optimizers هي طرق تُستخدم لتغيير سمات الشبكة العصبية مثل الوزن weight ومعدل التعلم learning rate.في التجربة الأولية، استخدمت مُحسِّن Adam لكل من الشبكات العصبية المولدة والمميزة ذات معدل التعلم، وbeta1 (معدل الانحلال الأسي second-moment estimates لتقديرات اللحظة الأولى exponential decay rate)، وbeta2 (معدل الانحلال الأسي exponential decay rate لتقديرات اللحظة الثانية second-moment estimates) والمعلمات الفائقة hyperparameters.

```
# Create optimizers for the discriminator D and generator G
# Define your learning rate
lr=0.0002
opt_d = optim.Adam(D.parameters(), lr=lr, betas=(0.5,
0.999))
opt_g = optim.Adam(G.parameters(), lr=lr, betas=(0.5,
0.999))
```

لقد وجدت أن النموذج يحقق أفضل أداء بقيم 0.0002، 0.5، 0.999 لمعدل التعلم، وbeta1، وbeta2 للمعلمات الفائقة على التوالي. لقد استخدمت حجم دفعة يبلغ 128 و40 فترة (تكرارات) لتدريب النموذج.

يمكنك ضبط النموذج باستخدام قيم معلمات فائقة مختلفة.

عملية التدريب

تدريب الشبكات العصبية عن طريق تغذية النماذج بالصور المعالجة مسبقًا. لقد استخدمت 40 تكرارًا في مراحل التدريب.

تدريب المميز

النماذج المدربة بدلا من ذلك، بدءا من المميز. قم بتغذية الصور الحقيقية للمميز وحساب الخطأ الحقيقي real loss.

ثم قم بإنشاء صور مزيفة عن طريق تغذية المولد بالضوضاء ثم قم بتغذية مخرجات المولد إلى المميز وحساب الخطأ المزيف fake loss.

الخطأ الإجمالي للمميز تقاس على النحو التالي،

$$Total\ loss = real\ loss + fake\ loss$$

الخطأ الإجمالي تتكون من خطأين. الأول هو اكتشاف الصور الحقيقية على أنها حقيقية والثاني هو اكتشاف الصور المزيفة على أنها مزيفة. تم حساب الدرجات الحقيقية والمزيفة باستخدام متوسط مخرجات المميز.

```
loss_fn = torch.nn.MSELoss()
def Real_loss(preds,targets):
    '''
       Calculates how close discriminator outputs are to
being real.
       param, D_out: discriminator logits
       return: real loss
    '''
    beta_distr =
torch.distributions.beta.Beta(1,5,validate_args=None)
    label_noise =
beta_distr.sample(sample_shape=targets.shape).to(torch.devic
e(device))
    loss= loss_fn(targets,preds-label_noise)
    return loss

def Fake_loss(preds,targets):
```

```
    '''
       Calculates how close discriminator outputs are to
being fake.
       param, D_out: discriminator logits
       return: fake loss
    '''
    beta_distr =
torch.distributions.beta.Beta(1,5,validate_args=None)
    label_noise =
beta_distr.sample(sample_shape=targets.shape).to(torch.devic
e(device))
    loss= loss_fn(targets,preds+label_noise)
    return loss
```

تدريب المولد

لتدريب المولد، قم بإنشاء صور مزيفة عن طريق تغذية المولد بالضوضاء. ثم حاول خداع المُميِّز باستخدام دالة الخطأ الحقيقية (نفس دالة الخطأ الحقيقية التي استخدمناها لتدريب المُميِّز).

حفظ الصور

احفظ الصور التي تم إنشاؤها ونماذج المولدات والمميز (إذا كنت تريد)في كل فترة تدريب. تم حفظ خطأ المولد وخطأ المميز والنتيجة الحقيقية والنتيجة المزيفةفي كل فترة وأخيرًا رسم خطأ المولد وخطأ المميزفي نفس الرسم البياني.

```
##Define your save path.
sample_dir = 'generated'
os.makedirs(sample_dir, exist_ok=True)
def save_samples(index, latent_tensors, generator,
show=True):
    fake_images = generator(latent_tensors)
    fake_fname = 'generated-images-
{0:0=4d}.png'.format(index)
    save_image(denorm(fake_images), os.path.join(sample_dir,
fake_fname), nrow=8)
    print('Saving', fake_fname)
    if show:
        fig, ax = plt.subplots(figsize=(8, 8))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(fake_images.cpu().detach(),
nrow=8).permute(1, 2, 0))
        plt.show()
fixed_latent = torch.randn(64, latent_size, 1, 1,
device=device)

losses_g = []
losses_d = []
real_scores = []
fake_scores = []
```

```
def train(D, G, d_optimizer, g_optimizer, epochs=1):
    iter_count = 0
    start_idx=1
    for epoch in range(epochs):
        for real_images in tqdm(trainloader):
            real_images=real_images.to(device)

            # Pass real images through discriminator
            D_out_real = D(real_images)
            label_real = torch.full(D_out_real.shape,
1.0).to(torch.device(device))
            real_loss = Real_loss(label_real,D_out_real)
            real_score = torch.mean(D_out_real).item()

            # Generate fake images
            noise = torch.randn(batch_size, latent_size, 1,
1).to(torch.device(device))
            fake_images =  G(noise)

            # Pass fake images through discriminator
            D_out_fake = D(fake_images)
            label_fake =  torch.full(D_out_fake.shape,
0).to(torch.device(device))
            fake_loss = Fake_loss(label_fake, D_out_fake)
            fake_score = torch.mean(D_out_fake).item()

            # Update discriminator weights
            loss_d = real_loss + fake_loss

            d_optimizer.zero_grad()
            loss_d.backward(retain_graph = True)
            d_optimizer.step()

            # Generate fake images
            noise2 = torch.randn(batch_size, latent_size, 1,
1).to(torch.device(device))
            fake_images2 =  G(noise2)

            gen_steps = 1
            for i in range(0, gen_steps ):
            # Try to fool the discriminator
                D_out_fake2 = D(fake_images2)

                # The label is set to 1(real-like) to fool
the discriminator
                label_real1 = torch.full(D_out_fake2.shape,
1.0).to(torch.device(device))
                loss_g = Real_loss(label_real1, D_out_fake2)

                # Update generator weights
```

```
                g_optimizer.zero_grad()
                loss_g.backward(retain_graph = (i<gen_steps
-1 ))
                g_optimizer.step()


        losses_g.append(loss_g.item())
        losses_d.append(loss_d.item())
        real_scores.append(real_score)
        fake_scores.append(fake_score)
        # Log losses & scores (last batch)
        print("Epoch [{}/{}], loss_g: {:.4f}, loss_d:
{:.4f}, real_score: {:.4f}, fake_score: {:.4f}".format(
        epoch+1, epochs, loss_g, loss_d, real_score,
fake_score))

        # Save generated images
        save_samples(epoch+start_idx, fixed_latent,G,
show=True)

        state_dis = {'dis_model': D.state_dict(), 'epoch':
epoch}
        state_gen = {'gen_model': G.state_dict(), 'epoch':
epoch}
        if not os.path.isdir('checkpoint'):
            os.mkdir('checkpoint')
        torch.save(state_dis,
'checkpoint/'+'D__'+str(epoch+1)) #each epoch
        torch.save(state_gen,
'checkpoint/'+'G__'+str(epoch+1)) #each epoch
#Train the GAN
train(D,G,opt_d,opt_g,epochs=15)
```

```
## Visualize your loss curve of D and G
fig, ax = plt.subplots()
plt.plot(losses_g, label='Discriminator', alpha=0.5)
plt.plot(losses_d, label='Generator', alpha=0.5)
plt.title("Training Losses")
plt.legend()
```

بعد انتهاء التكرارات التدريبية الأولى أو الثانية، سترى وجوه الأنمي تظهرفي النتائج. سيؤدي كل تكرار إلى تحسين أداء النموذج.

الصور المولدة

هذه بعض الصور التي تم إنشاؤها من نموذج GAN الخاص بي بعد 40 فترة. قريبة جدا أليس كذلك؟ يمكنك تحسين هذه الصور عن طريق ضبط معلمات النموذج.



هذا كل شيء!!!

لقد قمت الآن بإنشاء شخصيات الرسوم المتحركة (الانمي) الخاصة بك!!!

يمكنك العثور على الكود الكامل هنا.

المصدر:

https://aihalapathirana.medium.com/generative-adversarial-networks-for-anime-face-generation-pytorch-1b4037930e21

9) انشاء وجه مزيف باستخدام شبكة الخصومة التوليدية

Fake Face Generation Using GAN

في المقالة التالية، سنقوم بتعريف وتدريب نموذج شبكة الخصومة التوليدية الالتفافية العميقة Deep Convolutional Generative Adversarial Network(DCGAN) على مجموعة بيانات من الوجوه. الهدف الرئيسي من النموذج هو جعل شبكة المولدات تولد صورًا جديدة لوجوه بشرية مزيفة تبدو واقعية قدر الإمكان.

للقيام بذلك، سنحاول أولاً فهم الحدس وراء عمل شبكات GAN وشبكات DCGAN ثم دمج هذه المعرفة لبناء نموذج مولد الوجه المزيف Fake Face Generator Mode. بحلول نهاية هذه المقالة، ستكون قادرًا على إنشاء عينات مزيفة على أي مجموعة بيانات معينة، باستخدام المفاهيم الواردةفي هذه المقالة.

مقدمة

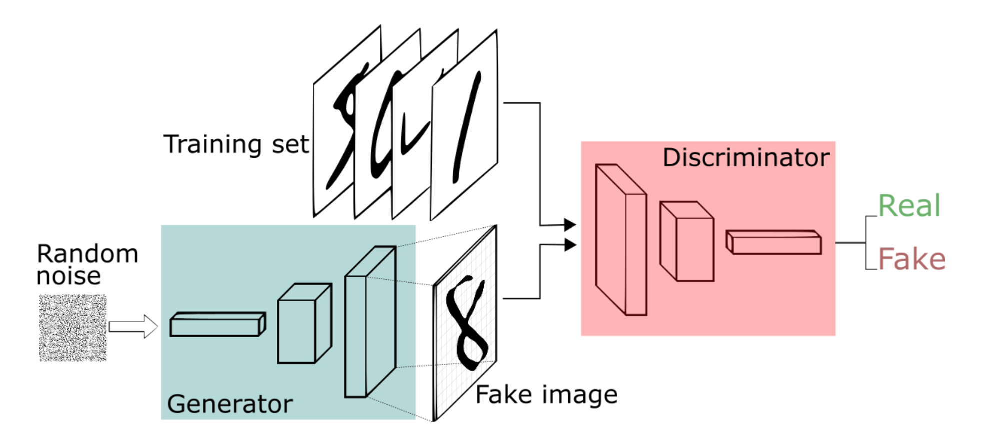
المقالة التالية مقسمة إلى قسمين:

- **النظرية**: فهم الحدس وراء عمل شبكات GAN و DCGAN.
- **التطبيق**: تنفيذ مولد الوجه المزيف في Pytorch.

هذه المقالة سوف تغطي كلا القسمين. لذلك دعونا نبدأ الرحلة....

النظري

الحدس وراء شبكات الخصومة التوليدية (GANs)



معمارية شبكة الخصومة التوليدية (GAN).

التعريف

يمكن تعريف شبكات GAN بشكل عام على أنها نموذج توليدي generative model يتيح لنا إنشاء صورة كاملة بالتوازي. إلى جانب عدة أنواع أخرى من النماذج التوليدية، تستخدم شبكات GAN دالة قابلة للتفاضل differentiable function تمثلها شبكة عصبية كشبكة المولد Generator Network.

شبكة المولد

تأخذ شبكة المولد Generator Network الضوضاء العشوائية random noise كمدخل، ثم تقوم بتشغيل الضوضاء من خلال الدالة القابلة للتفاضل (الشبكة العصبية) لتحويل الضوضاء وإعادة تشكيلها للحصول على معمارية يمكن التعرف عليها مشابهة للصور الموجودةفي مجموعة بيانات التدريب. يتم تحديد إخراج المولد من خلال اختيار الضوضاء العشوائية المدخلة. يؤدي تشغيل شبكة المولدات عبر عدة ضوضاء مدخلات عشوائية مختلفة إلى الحصول على صور مخرجات واقعية مختلفة.

الهدف النهائي للمولد هو تعلم توزيع مشابه لتوزيع مجموعة بيانات التدريب لأخذ عينات من الصور الواقعية. ولكي تتمكن من القيام بذلك، تحتاج شبكة المولدات إلى التدريب. تختلف عملية تدريب شبكات GAN كثيرًا، مقارنة بالنماذج التوليدية الأخرى (يتم تدريب معظم النماذج التوليدية عن طريق ضبط المعلمات لتعظيم احتمالية قيام المولد بإنشاء عينات واقعية. على سبيل المثال، شبكات الترميز التلقائي المتغيرة Variational Auto-Encoders(VAE). من ناحية أخرى، تستخدم شبكات GAN شبكة ثانية لتدريب المولد، تسمى شبكة المميز Discriminator Network.

شبكة المميز

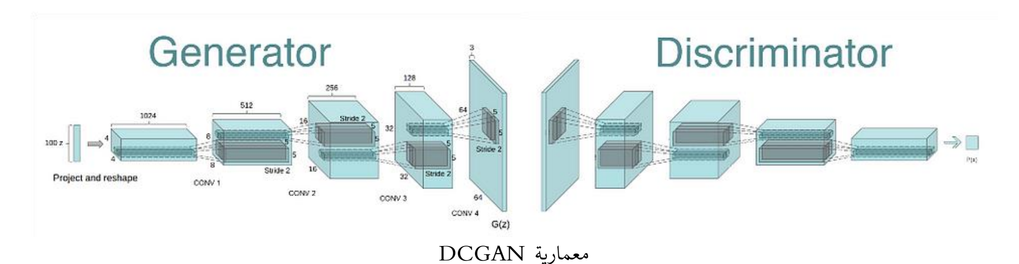
شبكة المميز Discriminator Network هي شبكة تصنيف أساسية تنتج احتمالية أن تكون الصورة حقيقية. لذلك، أثناء عملية التدريب، يتم عرض صور حقيقية من مجموعة التدريب على شبكة المميز في نصف الوقت وصور مزيفة من المولدفي نصف الوقت الآخر. هدف المميز هو تعيين احتمال قريب من 1 للصور الحقيقية واحتمال قريب من 0 للصور المزيفة.

من ناحية أخرى، يحاول المولد العكس، فهدفه هو إنشاء صور مزيفة، حيث يؤدي المميز إلى احتمال قريب من 1 (معتبرًا أنها صور حقيقية من مجموعة التدريب). ومع استمرار التدريب، سيصبح المميز أفضل في تصنيف الصور الحقيقية والمزيفة. لذا، لخداع المميز، سيضطر المولد إلى التحسين لإنتاج عينات أكثر واقعية. لذلك يمكننا أن نقول أن:

أي مُحسن سيتم اختياره؟ كيف يتم تحديد دالة التكلفة؟ ما المدة التي تحتاجها الشبكة للتدريب؟ وغيرها الكثير، والتي سيتم تناولها في القسم العملي.

العملي

يتم تقسيم جزء التنفيذ إلى سلسلة من المهام بدءًا من تحميل البيانات وحتى تحديد وتدريب شبكات الخصومة.في نهاية هذا القسم، ستتمكن من تصور نتائج المولد الذي تم تدريبه لمعرفة كيفية أدائه؛ يجب أن تبدو العينات التي تم إنشاؤها مثل الوجوه الواقعية إلى حد ما مع كميات صغيرة من الضوضاء.



معمارية DCGAN

1) الحصول على البيانات

ستستخدم مجموعة بيانات CelebFaces Attributes (CelebA) لتدريب شبكات الخصومة الخاصة بك. هذه البيانات عبارة عن مجموعة بيانات أكثر تعقيداً مقارنةً بـ MNIST. لذلك، نحن بحاجة إلى تحديد شبكة أعمق (DCGAN) لتحقيق نتائج جيدة. أود أن أقترح عليك استخدام GPU لأغراض التدريب.

2) إعداد البيانات

نظرًا لأن الهدف الرئيسي من هذه المقالة هو بناء نموذج DCGAN، فبدلاً من إجراء المعالجة المسبقة بأنفسنا، سنستخدم مجموعة بيانات تمت معالجتها مسبقًا. يمكنك تنزيل المجموعة الفرعية الأصغر من مجموعة بيانات CelebA من هنا. وإذا كنت مهتمًا بإجراء المعالجة المسبقة، فقم بما يلي:

- قم بقص الصور لإزالة الجزء الذي لا يشمل الوجه.
- قم بتغيير حجمها إلى صور NumPy بحجم 64x64x3.

الآن، سنقوم بإنشاء DataLoader للوصول إلى الصور على دفعات.

```
def get_dataloader(batch_size, image_size,
data_dir='train/'):
    """
    Batch the neural network data using DataLoader
    :param batch_size: The size of each batch; the number of
images in a batch
    :param img_size: The square size of the image data (x,
y)
    :param data_dir: Directory where image data is located
    :return: DataLoader with batched data
    """
```

```
    transform =
transforms.Compose([transforms.Resize(image_size),transforms
.CenterCrop(image_size),transforms.ToTensor()])

    dataset = datasets.ImageFolder(data_dir,transform =
transform)

    dataloader = torch.utils.data.DataLoader(dataset =
dataset,batch_size = batch_size,shuffle = True)
    return dataloader# Define function hyperparameters
batch_size = 256
img_size = 32# Call your function and get a dataloader
celeba_train_loader = get_dataloader(batch_size, img_size)
```

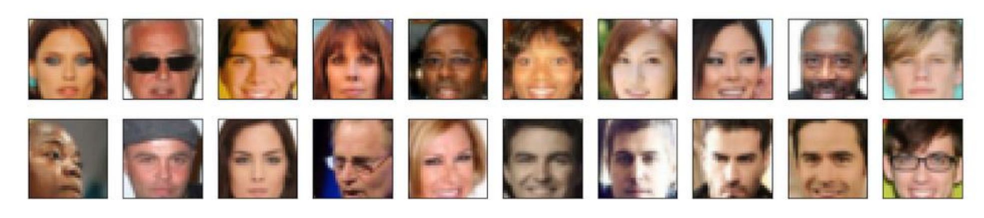
معلمات DataLoader الفائقة:

- يمكنك تحديد أي معلمة Batch_size معقولة.
- ومع ذلك، يجب أن يكون حجم الصورة 32. سيؤدي تغيير حجم البيانات إلى حجم أصغر إلى التدريب بشكل أسرع، مع الاستمرارفي إنشاء صور مقنعة للوجوه.

بعد ذلك، سنكتب بعض التعليمات البرمجية للحصول على تمثيل مرئي لمجموعة البيانات.

```
def imshow(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))# obtain one
batch of training images
dataiter = iter(celeba_train_loader)
images, _ = dataiter.next() # _ for no labels# plot the
images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(20, 4))
plot_size=20
for idx in np.arange(plot_size):
    ax = fig.add_subplot(2, plot_size/2, idx+1, xticks=[],
yticks=[])
    imshow(images[idx])
```

ضعفي اعتبارك تحويل صور Tensor إلى نوع NumPy وتبديل الأبعاد لعرض الصورة بشكل صحيح بناءً على الكود أعلاه (في Dataloader قمنا بتحويل الصور إلى Tensor). قم بتشغيل هذا الجزء من التعليمات البرمجية للحصول على تصور لمجموعة البيانات.



الصور تتمحور حول الوجوه

الآن قبل البدء بالقسم التالي (تعريف النموذج Defining Model)، سنكتب دالة لتحجيم scale بيانات الصورة إلى نطاق بكسل من ـ1 إلى 1 والذي سنستخدمه أثناء التدريب. السبب وراء القيام بذلك هو أن مخرجات المولد المنشط ستحتوي على قيم بكسل في نطاق من ـ1 إلى 1، ولذا، نحتاج إلى إعادة قياس صور التدريب الخاصة بنا إلى نطاق من ـ1 إلى 1 (في الوقت الحالي، هم تقع في النطاق 0ـ1).

```
def scale(x, feature_range=(-1, 1)):
''' Scale takes in an image x and returns that image, scaled
with a feature_range of pixel values from -1 to 1.
This function assumes that the input x is already scaled
from 0-1.'''
# assume x is scaled to (0, 1)
# scale to feature_range and return scaled x
min, max = feature_range
x = x*(max-min) + min
return x
```

3) تعريف النموذج

تتكون الشبكة GAN من شبكتين متعارضتين، شبكة المميز والمولد. لذلك،في هذا القسم، سوف نقوم بتعريف معمارية كل منهما.

المميز

هذا مصنف تلافيفي convolutional classifier، فقط بدون أي طبقات MaxpPooling. هنا هو كود شبكة المميز.

```
def conv(input_c,output,kernel_size,stride = 2,padding  = 1,
batch_norm = True):
    layers =[]
    con =
nn.Conv2d(input_c,output,kernel_size,stride,padding,bias =
False)
    layers.append(con)

    if batch_norm:
        layers.append(nn.BatchNorm2d(output))

    return nn.Sequential(*layers)class
Discriminator(nn.Module):def __init__(self, conv_dim):
        """
        Initialize the Discriminator Module
        :param conv_dim: The depth of the first
convolutional layer
        """
        #complete init functionsuper(Discriminator,
self).__init__()
```

```
        self.conv_dim = conv_dim
        self.layer_1 = conv(3,conv_dim,4,batch_norm = False)
#16
        self.layer_2 = conv(conv_dim,conv_dim*2,4) #8
        self.layer_3 = conv(conv_dim*2,conv_dim*4,4) #4
        self.fc = nn.Linear(conv_dim*4*4*4,1)def
forward(self, x):
        """
        Forward propagation of the neural network
        :param x: The input to the neural network
        :return: Discriminator logits; the output of the
neural network
        """
        # define feedforward behavior
        x = F.leaky_relu(self.layer_1(x))
        x = F.leaky_relu(self.layer_2(x))
        x = F.leaky_relu(self.layer_3(x))
        x = x.view(-1,self.conv_dim*4*4*4)
        x = self.fc(x)
        return x
```

الشرح:

- تتكون المعمارية التالية من ثلاث طبقات تلافيفية وطبقة نهائية متصلة بالكامل، والتي تنتج لوجيتاً logit واحداً. يحدد هذا اللوغاريتم ما إذا كانت الصورة حقيقية أم لا.
- كل طبقة تلافيفية، باستثناء الطبقة الأولى، يتبعها التسوية بالدفعات Batch Normalization (محددةفي دالة مساعد conv helper function).
- بالنسبة للوحدات المخفية، استخدمنا دالة تنشيط ReLU المتسربة كما تمت مناقشتهافي القسم النظري.
- بعد كل طبقة التفافية، يصبح الارتفاع والعرض إلى النصف. على سبيل المثال، بعد عملية الالتفاف الأولى، سيتم تغيير حجم الصور مقاس 32X32 إلى 16X16 وهكذا.

يمكن حساب البعد الناتج باستخدام الصيغة التالية:

$$O = \frac{W-K+2P}{S} + 1$$

حيث O هو ارتفاع/طول الإخراج، W هو ارتفاع/طول الإدخال، K هو حجم الفلتر، P هو الحشو padding، وS هي الخطوة stride.

- يعتمد عدد خرائط الميزات بعد كل التفاف على المعلمة conv_dim (في تطبيقي conv_dim = 64).

في تعريف النموذج هذا، لم نطبق دالة التنشيط السيني Sigmoid على لوجيت الإخراج النهائي final output logit. هذا بسبب اختيار دالة الخطأ لدينا. هنا بدلاً من استخدام BCE (Binary Cross-Entropy Loss) العادي، سنستخدم BCEWithLogitLoss، والذي يعتبر إصدارًا مستقرًا عدديًا من BCE. يتم تعريف BCEWithLogitLoss بحيث يقوم أولاً بتطبيق دالة التنشيط Sigmoid على اللوجيت ثم يحسب الخطأ، على عكس BCE. يمكنك قراءة المزيد عن دوال الخطأ هذه هنا.

المولد

يجب على المولد أن يقوم بتجميع المدخلات وإنشاء صورة جديدة بنفس حجم بيانات التدريب لدينا 32X32X3. للقيام بذلك سوف نستخدم طبقات تلافيفية منقولة. هنا هو كود شبكة المولدات.

```
def deconv(input_c,output,kernel_size,stride = 2, padding
=1, batch_norm = True):
    layers = []
    decon =
nn.ConvTranspose2d(input_c,output,kernel_size,stride,padding
,bias = False)
    layers.append(decon)

    if batch_norm:
        layers.append(nn.BatchNorm2d(output))
    return nn.Sequential(*layers)class Generator(nn.Module):

    def __init__(self, z_size, conv_dim):
        """
        Initialize the Generator Module
        :param z_size: The length of the input latent
vector, z
        :param conv_dim: The depth of the inputs to the
*last* transpose convolutional layer
        """
        super(Generator, self).__init__()
        # complete init function
        self.conv_dim = conv_dim
        self.fc = nn.Linear(z_size,conv_dim*8*2*2)
        self.layer_1 = deconv(conv_dim*8,conv_dim*4,4) #4
        self.layer_2 = deconv(conv_dim*4,conv_dim*2,4) #8
        self.layer_3 = deconv(conv_dim*2,conv_dim,4) #16
        self.layer_4 = deconv(conv_dim,3,4,batch_norm =
False) #32


    def forward(self, x):
        """
        Forward propagation of the neural network
        :param x: The input to the neural network
        :return: A 32x32x3 Tensor image as output
```

```
        """
        # define feedforward behavior
        x = self.fc(x)
        x = x.view(-1,self.conv_dim*8,2,2)
#(batch_size,depth,width,height)
        x = F.relu(self.layer_1(x))
        x = F.relu(self.layer_2(x))
        x = F.relu(self.layer_3(x))
        x = torch.tanh(self.layer_4(x))
        return x
```

الشرح:

- تتكون المعمارية التالية من طبقة متصلة بالكامل تليها أربع طبقات تلافيفية منقولة. يتم تعريف هذه المعمارية بحيث ينتج عن الإخراج بعد الطبقة التلافيفية الرابعة صورة ذات أبعاد 32X32X3 (حجم الصورة من مجموعة بيانات التدريب).
- المدخلات إلى المولد عبارة عن متجهات بطول z_size (z_size هو متجه الضوضاء noise vector).
- كل طبقة تلافيفية منقولة، باستثناء الطبقة الأخيرة، يتبعها تسوية بالدفعات (محددفي دالة المساعدة deconv).
- بالنسبة للوحدات المخفية، استخدمنا دالة التنشيط ReLU.
- بعد كل تبديل للطبقة التلافيفية، يصبح الارتفاع والعرض مضاعفين. على سبيل المثال، بعد الالتفاف المنقول الأول، سيتم تغيير حجم الصور 2X2إلى 4X4 وهكذا.

يمكن حسابها باستخدام الصيغة التالية:

```
# Padding==Same:
H = H1 * stride
# Padding==Valid
H = (H1-1) * stride + HF
```

حيث H = حجم الإخراج، H1 = حجم الإدخال، HF = حجم الفلتر.

يعتمد عدد خرائط الميزات بعد كل تحويل تبديل على المعلمة conv_dim (في تطبيقي conv_dim = 64).

(4) تهيئة أوزان الشبكة

للمساعدةفي تقارب النماذج، قمت بتهيئة أوزان الطبقات التلافيفية والخطيةفي النموذج بناءً على ورقة DCGAN الأصلية، والتي تقول: تتم تهيئة جميع الأوزان من توزيع عادي مركزه صفر مع انحراف معياري قدره 0.02.

```
def weights_init_normal(m):
"""
Applies initial weights to certain layers in a model .
The weights are taken from a normal distribution
with mean = 0, std dev = 0.02.
:param m: A module or layer in a network
"""
# classname will be something like:
# `Conv`, `BatchNorm2d`, `Linear`, etc.
classname = m.__class__.__name__

if hasattr(m,'weight') and (classname.find('Conv') != -1 or
classname.find('Linear') != -1):

m.weight.data.normal_(0.0,0.02)

if hasattr(m,'bias') and m.bias is not None:
m.bias.data.zero_()
```

سيؤدي هذا إلى تهيئة الأوزان للتوزيع الطبيعي، المتمركز حول 0، مع انحراف معياري قدره 0.02.

قد يتم ترك مصطلحات التحيز bias، إذا كانت موجودة، بمفردها أو تعيينها على 0.

5) بناء شبكة كاملة

حدد المعلمات الفائقة hyperparameters للنماذج الخاصة بك وقم بإنشاء مثيل للمميز والمولد من الفئات المحددةفي قسم تعريف النموذج. هنا الكود.

```
def build_network(d_conv_dim, g_conv_dim, z_size):
    # define discriminator and generator
    D = Discriminator(d_conv_dim)
    G = Generator(z_size=z_size, conv_dim=g_conv_dim)#
initialize model weights
    D.apply(weights_init_normal)
    G.apply(weights_init_normal)print(D)
    print()
    print(G)

    return D, G

# Define model hyperparams
d_conv_dim = 64
g_conv_dim = 64
z_size = 100D, G = build_network(d_conv_dim, g_conv_dim,
z_size)
```

عند تشغيل الكود أعلاه تحصل على الإخراج التالي. كما يصف أيضًا معمارية النموذج لنماذج المميز والمولد.

```
Discriminator(
  (layer_1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
  (layer_2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (layer_3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (fc): Linear(in_features=4096, out_features=1, bias=True)
)

Generator(
  (fc): Linear(in_features=100, out_features=2048, bias=True)
  (layer_1): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (layer_2): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (layer_3): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (layer_4): Sequential(
    (0): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
)
```

نماذج التمييز والمولدات التي تم إنشاؤها

(6) عملية التدريب

تشتمل عملية التدريب على تحديد دوال الخطأ واختيار المحسن وأخيراً تدريب النموذج.

خطأ المميز والمولد

خطأ المميز

بالنسبة للمميز، الخطأ الإجمالي هي مجموع (d_real_loss + d_fake_loss)، حيث d_real_loss هي الخطأ التي تم الحصول عليها على الصور من بيانات التدريب وd_fake_loss هي الخطأ التي تم الحصول عليها على الصور التي تم إنشاؤها من شبكة المولد. على سبيل المثال:

z: متجه الضوضاء.

i: صورة من مجموعة التدريب.

G(z): الصورة المولدة.

D(G(z)): إخراج المميز على الصورة التي تم إنشاؤها.

D(i): إخراج المُميِّز على صورة مجموعة بيانات التدريب.

Loss = real_loss(D(i)) + fake_loss(D(G(z)))

- تذكر أننا نريد من المميز أن يُخرج 1 للصور الحقيقية و0 للصور المزيفة، لذلك نحتاج إلى إعداد الاخطاء لتعكس ذلك (ضع هذا السطرفي الاعتبار أثناء قراءة الكود أدناه).

خطأ المولد

سيبدو خطأ المولد متشابهة فقط مع التسميات المقلوبة flipped labels. هدف المولد هو جعل المُميز يعتقد أن الصور التي تم إنشاؤها حقيقية. على سبيل المثال:

z: متجه الضوضاء.

G(z): الصورة المولدة.

D(G(z)): إخراج المميز على الصورة التي تم إنشاؤها.

Loss = real_loss(D(G(z)))

إليك كود real_loss وfake_loss.

```
def real_loss(D_out):
    '''Calculates how close discriminator outputs are to
being real.
       param, D_out: discriminator logits
       return: real loss'''
    batch_size = D_out.size(0)
    labels = torch.ones(batch_size)
    if train_on_gpu:
        labels = labels.cuda()
    criterion = nn.BCEWithLogitsLoss()
    loss = criterion(D_out.squeeze(),labels)
    return lossdef fake_loss(D_out):
    '''Calculates how close discriminator outputs are to
being fake.
       param, D_out: discriminator logits
       return: fake loss'''
    batch_size = D_out.size(0)
    labels = torch.zeros(batch_size)
    if train_on_gpu:
        labels = labels.cuda()
    criterion =  nn.BCEWithLogitsLoss()
    loss = criterion(D_out.squeeze(),labels)
    return loss
```

المحسنات

بالنسبة لشبكات GAN، نحدد محسنين optimizers، أحدهما للمولد والآخر للمميز. والفكرة هي تشغيلهمافي وقت واحد لمواصلة تحسين الشبكتين.في هذا التنفيذ، استخدمت محسن Adamفي كلتا الحالتين. لمعرفة المزيد عن أدوات تحسين الأداء المختلفة، راجع هذا الرابط.

```
# Create optimizers for the discriminator D and generator G
d_optimizer = optim.Adam(D.parameters(),lr = .0002, betas =
[0.5,0.999])
g_optimizer = optim.Adam(G.parameters(),lr = .0002, betas =
[0.5,0.999])
```

يعتمد معدل التعلم (lr) وقيم بيتا على ورقة DCGAN الأصلية.

التدريب

سيتضمن التدريب التناوب بين تدريب المُميز والمولد. سنستخدم دالتي real_loss و fake_loss المحددتين سابقًا لمساعدتنافي حساب اخطاء المميز والمولد.

يجب عليك تدريب الشخص الذي يقوم بالتمييز بالتناوب بين الصور الحقيقية والمزيفة.

ثم المولد الذي يحاول خداع المُميِّز ويجب أن يكون له دالة خطأ معاكسة.

هنا هو كود للتدريب.

```
def train(D, G, n_epochs, print_every=50):
    '''Trains adversarial networks for some number of epochs
       param, D: the discriminator network
       param, G: the generator network
       param, n_epochs: number of epochs to train for
       param, print_every: when to print and record the
models' losses
       return: D and G losses'''

    # move models to GPU
    if train_on_gpu:
        D.cuda()
        G.cuda()# keep track of loss and generated, "fake"
samples
    samples = []
    losses = []# Get some fixed data for sampling. These are
images that are held
    # constant throughout training, and allow us to inspect
the model's performance
    sample_size=16
    fixed_z = np.random.uniform(-1, 1, size=(sample_size,
z_size))
    fixed_z = torch.from_numpy(fixed_z).float()
```

```
    # move z to GPU if available
    if train_on_gpu:
        fixed_z = fixed_z.cuda()# epoch training loop
    for epoch in range(n_epochs):# batch training loop
        for batch_i, (real_images, _) in
enumerate(celeba_train_loader):batch_size =
real_images.size(0)
            real_images = scale(real_images)
            if train_on_gpu:
                real_images = real_images.cuda()

            # 1. Train the discriminator on real and fake
ima.ges
            d_optimizer.zero_grad()
            d_out_real = D(real_images)
            z = np.random.uniform(-1,1,size =
(batch_size,z_size))
            z = torch.from_numpy(z).float()
            if train_on_gpu:
                z = z.cuda()
            d_loss = real_loss(d_out_real) +
fake_loss(D(G(z)))
            d_loss.backward()
            d_optimizer.step()
            # 2. Train the generator with an adversarial
loss
            G.train()
            g_optimizer.zero_grad()
            z = np.random.uniform(-1,1,size =
(batch_size,z_size))
            z = torch.from_numpy(z).float()
            if train_on_gpu:
                z = z.cuda()
            g_loss = real_loss(D(G(z)))
            g_loss.backward()
            g_optimizer.step()

            # Print some loss stats
            if batch_i % print_every == 0:
                # append discriminator loss and generator
loss
                losses.append((d_loss.item(),
g_loss.item()))
                # print discriminator and generator loss
                print('Epoch [{:5d}/{:5d}] | d_loss: {:6.4f}
| g_loss: {:6.4f}'.format(
                        epoch+1, n_epochs, d_loss.item(),
g_loss.item()))## AFTER EACH EPOCH##
        # this code assumes your generator is named G, feel
free to change the name
```

```
        # generate and save sample, fake images
        G.eval() # for generating samples
        samples_z = G(fixed_z)
        samples.append(samples_z)
        G.train() # back to training mode# Save training
generator samples
    with open('train_samples.pkl', 'wb') as f:
        pkl.dump(samples, f)

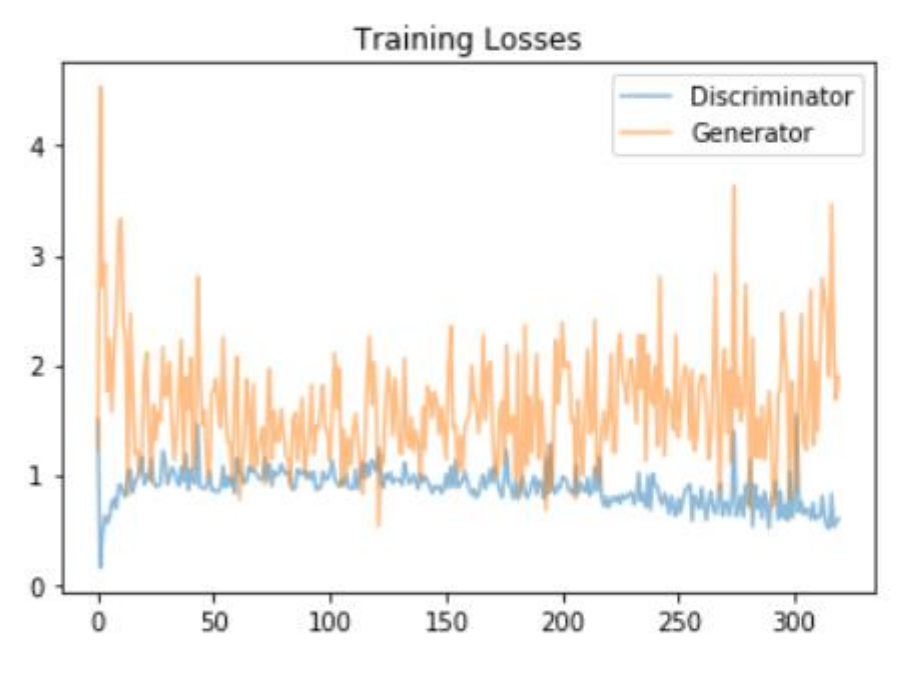
    # finally return losses
    return losses


# set number of epochs
n_epochs = 40# call training function
losses = train(D, G, n_epochs=n_epochs)
```

يتم تنفيذ التدريب على مدى 40 فترة epochs باستخدام وحدة معالجة الرسوميات GPU، ولهذا السبب اضطررت إلى نقل النماذج والمدخلات الخاصة بي من وحدة المعالجة المركزية CPU إلى وحدة معالجة الرسوميات GPU.

(7)النتائج

فيما يلي مخطط اخطاء التدريب للمولد والمميز المسجل بعد كل فترة.



خطأ التدريب للمميز والمولد

يرجع التقلب الكبيرفي خطأ تدريب المولد إلى أن المدخلات إلى شبكة المولدات عبارة عن مجموعة من متجهات الضوضاء العشوائية (كل منها بحجم z_size)، يتم أخذ كل عينة من توزيع موحد قدره (ــ1,1) لإنشاء صور جديدة لكل فترة.

في مخطط المميز، يمكننا ملاحظة ارتفاعفي خطأ التدريب (حوالي 50 على المحور السيني) يليه انخفاض تدريجي حتى النهاية، وذلك لأن المولد بدأفي توليد بعض الصور الواقعية التي خدعت المميز، مما أدى إلى لزيادة الخطأ. ولكن ببطء مع تقدم التدريب، يصبح المميز أفضل في تصنيف الصور المزيفة والحقيقية، مما يؤدي إلى انخفاض تدريجي في أخطاء التدريب.

- العينات التي تم إنشاؤها بعد 40 فترة.



إنشاء صور مزيفة

تمكن نموذجنا من إنتاج صور جديدة لوجوه بشرية مزيفة تبدو واقعية قدر الإمكان. يمكننا أيضًا أن نلاحظ أن جميع الصور تكون أفتحفي الظل، حتى الوجوه البنية تكون أفتح قليلاً. وذلك لأن مجموعة بيانات CelebA متحيزة؛ وتتكون من وجوه "المشاهير celebrity" ومعظمها بيضاء. ومع ذلك، نجح DCGANفي إنشاء صور شبه حقيقية من مجرد ضوضاء.

رابط الكود:

https://github.com/vjrahil/Face-Generator

المصدر:
https://towardsdatascience.com/fake-face-generator-using-dcgan-model-ae9322ccfd65

10) توليد وجه إنساني باستخدام شبكات الخصومة التوليدية
Generating Human Face using GAN

في هذا المشروع، سأوضح كيفية إنشاء وجوه بشرية human faces باستخدام شبكة الخصومة التوليدية Generative Adversarial Network (GAN)، والتي ربما لا تكون موجودةفي الحياة الواقعية.

شبكات الخصومة التوليدية الالتفافية العميقة (DC-GAN)

سأستخدم شبكات الخصومة التوليدية الالتفافية العميقة Deep Convolution Generative Adversarial Network (DC-GAN) لهذه المهمة. أنا أستخدم مجموعة بيانات CelebA لتدريب الشبكة. تحتوي مجموعة البيانات هذه على 2,00,000 صورة لأشخاص معروفين. أفترض أن لديك فهمًا نظريًا لشبكات GAN. سأستخدم إطار عمل TensorFlow في هذا البرنامج التعليمي. هيا نبدأ.

هذه هي الطريقة التي سيبدو بها سير العمل لدينا:

1) تسوية الصور.
2) إنشاء شبكة المولدات والمميزات.
3) تدريب الشبكة وتوليد وجوه جديدة.

فيما يلي بعض الصور من مجموعة البيانات لدينا.



تسوية الصور

- كخطوة أولى، نقوم باستيراد المكتبات التي سنستفيد منها.
- نقوم بتحميل جميع الصور باستخدام PIL. أثناء تحميل الصور نقوم بقص جميع الصور حول الوجه وتغيير حجمها إلى (64, 64, 3).
- وتقع هذه الصورفي حدود (0, 255). نقوم بسحق نطاق البت لهذه الصور بين (−1, 1)، والذي يقعفي نطاق تنشيط tanh.

```
import glob
import numpy as np
from PIL import Image
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import reduce_mean
from tensorflow.train import AdamOptimizer as adam
from tensorflow.nn import sigmoid_cross_entropy_with_logits
as loss
from tensorflow.layers import dense, batch_normalization,
conv2d_transpose, conv2d

image_ids = glob.glob('../input/data/*')

crop = (30, 55, 150, 175)
images =
[np.array((Image.open(i).crop(crop)).resize((64,64))) for i
in image_ids]

for i in range(len(images)):
    images[i] = ((images[i] - images[i].min())/(255 -
images[i].min()))
    images[i] = images[i]*2-1

images = np.array(images)
```

إنشاء الشبكة

- أقوم بتنفيذ DCGAN هنا لأكون محددًا. وهو ما يسمى بشبكات الخصومة التوليدية الالتفافية العميقة. هذا هو شكل مختلف من شبكة GAN القياسية التي تم تقديمهافي عام 2014 بواسطةIan Goodfellow Ian Goo. تستخدم DCGAN طبقات الالتفاف Convolution layers بدلاً من جميع الطبقات المتصلة بالكامل fully connected layers.
- المفهوم الكامن وراء GAN هو أن لديها شبكتين تسمى شبكة المولد Generator Discriminator. تتمثل مهمة المولد Generatorفي إنشاء صور واقعية المظهر من

الضوضاء وخداع المُميِّز Discriminator. ومن ناحية أخرى، فإن مهمة المميز هي التمييز بين الصور الحقيقية والمزيفة. يتم تدريب هاتين الشبكتين بشكل منفصل.

- في بداية كلتا الشبكتين كانتا ضعيفتينفي دوالهما ولكن مع استمرارنافي التدريب، يتحسن المميزفي التمييز بين الصور الحقيقية والمزيفة ويتحسن المولدفي توليد صورة ذات مظهر حقيقي بحيث يمكنه خداع المُميِّز. ومع ذلك، فإن مهمة تدريب GAN ليست سهلة للغاية.
- تتشابه معمارية الشبكات واختيار المعلمات الفائقة تماماً مع تلك المستخدمةفي ورقة DCGAN. وهي ورقة بحثية لطيفة جداً تشرح كل ما يتعلق بـ DCGAN، وأوصيك بشدة بإلقاء نظرة عليها.

```
def generator(noise, reuse=False, alpha=0.2, training=True):

    with tf.variable_scope('generator', reuse=reuse):

        x = dense(noise, 4*4*512)
        x = tf.reshape(x, (-1, 4, 4, 512))
        x = batch_normalization(x, training=training)
        x = tf.maximum(0., x)

        x = conv2d_transpose(x, 256, 5, 2, padding='same')
        x = batch_normalization(x, training=training)
        x = tf.maximum(0., x)

        x = conv2d_transpose(x, 128, 5, 2, padding='same')
        x = batch_normalization(x, training=training)
        x = tf.maximum(0., x)

        x = conv2d_transpose(x, 64, 5, 2, padding='same')
        x = batch_normalization(x, training=training)
        x = tf.maximum(0., x)

        logits = conv2d_transpose(x, 3, 5, 2,
padding='same')
        out = tf.tanh(logits)

        return out, logits
```

سنقوم بتمرير ضوضاء موزعة بشكل موحد uniformly distributed noise إلى المولد. يقوم المولد بتحويل هذا الضوضاء إلى صورة بحجم (64, 64, 3). نحن نستخدم تحويل الالتفافي هذه العملية. يتم استخدام طبقات التسوية بالدفعات Batch normalization layers بعد طبقة الالتفاف المنقولة transpose convolution layer بدلاً من الطبقة الأخيرة. نحن نستخدم تنشيط Relu بعد طبقات التسوية بالدفعات. نقوم بتمرير صورتنا النهائية من خلال تنشيط tanh لسحق نطاق البكسل بين (-1, 1).

```
def discriminator(x, reuse=False, alpha=0.2, training=True):

    with tf.variable_scope('discriminator', reuse=reuse):

        x = conv2d(x, 32, 5, 2, padding='same')
        x = tf.maximum(alpha*x, x)

        x = conv2d(x, 64, 5, 2, padding='same')
        x = batch_normalization(x, training=training)
        x = tf.maximum(alpha*x, x)

        x = conv2d(x, 128, 5, 2, padding='same')
        x = batch_normalization(x, training=training)
        x = tf.maximum(alpha*x, x)

        x = conv2d(x, 256, 5, 2, padding='same')
        x = batch_normalization(x, training=training)
        x = tf.maximum(alpha*x, x)

        flatten = tf.reshape(x, (-1, 4*4*256))
        logits = dense(flatten, 1)
        out = tf.sigmoid(logits)

        return out, logits
```

سنمرر موتر الشكل (64, 64, 3) إلى المُميِّز. يعطي المميز مخرجًا واحداً يوضح ما إذا كانت هذه الصورة حقيقية أم مزيفة. نقوم بتمرير الإخراج النهائي من خلال التنشيط sigmoid. الذي يسحق الإخراج بين (0، 1). إذا كان الإخراج قريبًا من 1 يعني أن المميز تحدد الصورة على أنها صورة حقيقية وإذا كان الإخراج قريبًا من 0، فسيتم تعريف الصورة على أنها صورة مزيفة.

```
def inputs(real_dim, noise_dim):

    inputs_real = tf.placeholder(tf.float32, (None,
*real_dim), name='input_real')
    inputs_noise = tf.placeholder(tf.float32, (None,
noise_dim), name='input_noise')
    return inputs_real, inputs_noise

# building the graph
tf.reset_default_graph()

input_real, input_noise = inputs(input_shape, noise_size)
gen_noise, gen_logits = generator(input_noise)
dis_out_real, dis_logits_real = discriminator(input_real)
dis_out_fake, dis_logits_fake = discriminator(gen_noise,
reuse=True)

# defining losses
shape = dis_logits_real
```

```
dis_loss_real = reduce_mean(loss(logits=dis_logits_real,
labels=tf.ones_like(shape*smooth)))
dis_loss_fake = reduce_mean(loss(logits=dis_logits_fake,
labels=tf.zeros_like(shape)))
gen_loss = reduce_mean(loss(logits=dis_logits_fake,
labels=tf.ones_like(shape*smooth)))
dis_loss = dis_loss_real + dis_loss_fake

# defining optimizers
total_vars = tf.trainable_variables()

dis_vars = [var for var in total_vars if var.name[0] == 'd']
gen_vars = [var for var in total_vars if var.name[0] == 'g']
dis_opt = adam(learning_rate=learning_rate,
beta1=beta1).minimize(dis_loss, var_list=dis_vars)
gen_opt = adam(learning_rate=learning_rate,
beta1=beta1).minimize(gen_loss, var_list=gen_vars)
```

يتم تعريف دالتين مختلفتين للخطأ للمولد والمميز. الشيء نفسه ينطبق على المحسن.

تدريب الشبكة

فيما يلي اختيار المعلمات الفائقة hyperparameters. معدل التعلم Learning-rate هو 0.0002، حجم الضوضاء size of the noise هو 100، عامل تنعيم التسمية label smoothing factor هو 0.9، معلمة التسرب leak parameter لـ LeakyRelu هي 0.2، beta1 لمحسن Adam هو 0.5.

```
# hyperparameters
beta1 = 0.5
alpha = 0.2
smooth = 0.9
noise_size = 100
learning_rate = 0.0002
input_shape = (64,64,3)
```

- حجم الدفعة Batch-size هو 128. لقد قمت بتدريب الشبكة لمدة 15 فترة epochs. فيما يلي كود التدريب.

```
batch_size = 128
epochs = 15
iters = len(image_ids)//batch_size
saver = tf.train.Saver(var_list = gen_vars)

with tf.Session() as sess:

    sess.run(tf.global_variables_initializer())

    for e in range(epochs):
```

```
        for i in range(iters-1):

            batch_images =
images[i*batch_size:(i+1)*batch_size]
            batch_noise = np.random.uniform(-1, 1,
size=(batch_size, noise_size))

            sess.run(dis_opt, feed_dict={input_real:
batch_images, input_noise: batch_noise})
            sess.run(gen_opt, feed_dict={input_real:
batch_images, input_noise: batch_noise})

            if i%50 == 0:
                print("Epoch {}/{}...".format(e+1, epochs),
"Batch No {}/{}".format(i+1, iters))

        loss_dis = sess.run(dis_loss, {input_noise:
batch_noise, input_real: batch_images})
        loss_gen = gen_loss.eval({input_real: batch_images,
input_noise: batch_noise})

        print("Epoch {}/{}...".format(e+1,
epochs),"Discriminator Loss: {:.4f}...".format(loss_dis),
              "Generator Loss: {:.4f}".format(loss_gen))

        sample_noise = np.random.uniform(-1, 1, size=(8,
noise_size))
        gen_samples = sess.run(generator(input_noise,
reuse=True, alpha=alpha),
                               feed_dict={input_noise:
sample_noise})

        view_samples(-1, gen_samples, 2, 4, (10,5))
        plt.show()
        saver.save(sess, './checkpoints/generator.ckpt')
```

فيما يلي الصور التي تم إنشاؤها بواسطة الشبكة بعد التدريب. يمكننا إنشاء صور أكثر واقعية من خلال جعل شبكتنا أعمق، لكن تدريب النموذج سيستغرق الكثير من الوقت. يمكننا أيضًا تعديل شبكتنا لإنتاج صور عالية الدقة ولكن مرة أخرى على حساب وقت التدريب.



المصادر:

- هذه مدونة تمهيدية رائعة عن GAN.
- يرجى الاطلاع على هذه الورقة المفيدة حقًا حول DCGAN.
- فيما يلي رابط Git-Hub لهذا المشروع.

المصدر:

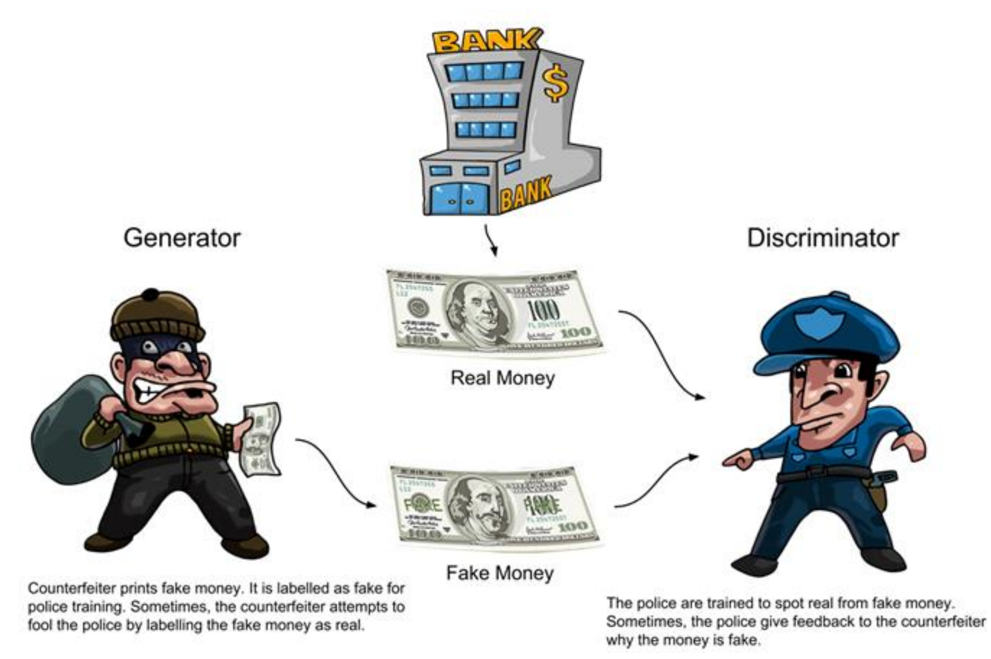
https://medium.com/@shiva-verma/generating-human-faces-using-adversarial-network-960863bc1deb

11) شيخوخة الوجه باستخدام شبكات الخصومة التوليدية Face Aging Using GANs

مقدمة

شبكات الخصومة التوليدية Generative Adversarial Networks هي نوع من معماريات الشبكات العصبية العميقة التي تستخدم التعلم الآلي غير الخاضع للأشراف لإنشاء البيانات. تم تقديمها في عام 2014،في ورقة بحثية كتبها إيان جودفيلو، ويوشوا بينجيو، وآرون كورفيل، والتي يمكن العثور عليها على الرابط التالي: https://arxiv.org/pdf/1406.2661.. لدى شبكات GAN العديد من التطبيقات، بمافي ذلك توليد الصور وتطوير الأدوية.

ستقدم لك هذه المدونة المكونات الأساسية لشبكات GAN. سوف يرشدك إلى كيفية عمل كل مكون والمفاهيم والتكنولوجيا المهمة وراء شبكات GAN. كما سيعطيك لمحة موجزة عن فوائد وعيوب استخدام شبكات GAN وإلقاء نظرة ثاقبة على بعض تطبيقات العالم الحقيقي. بعد فهم معمارية GAN، سنرى كيف يتم تطبيق GANفي مواجهة مشكلة الشيخوخة aging problem.



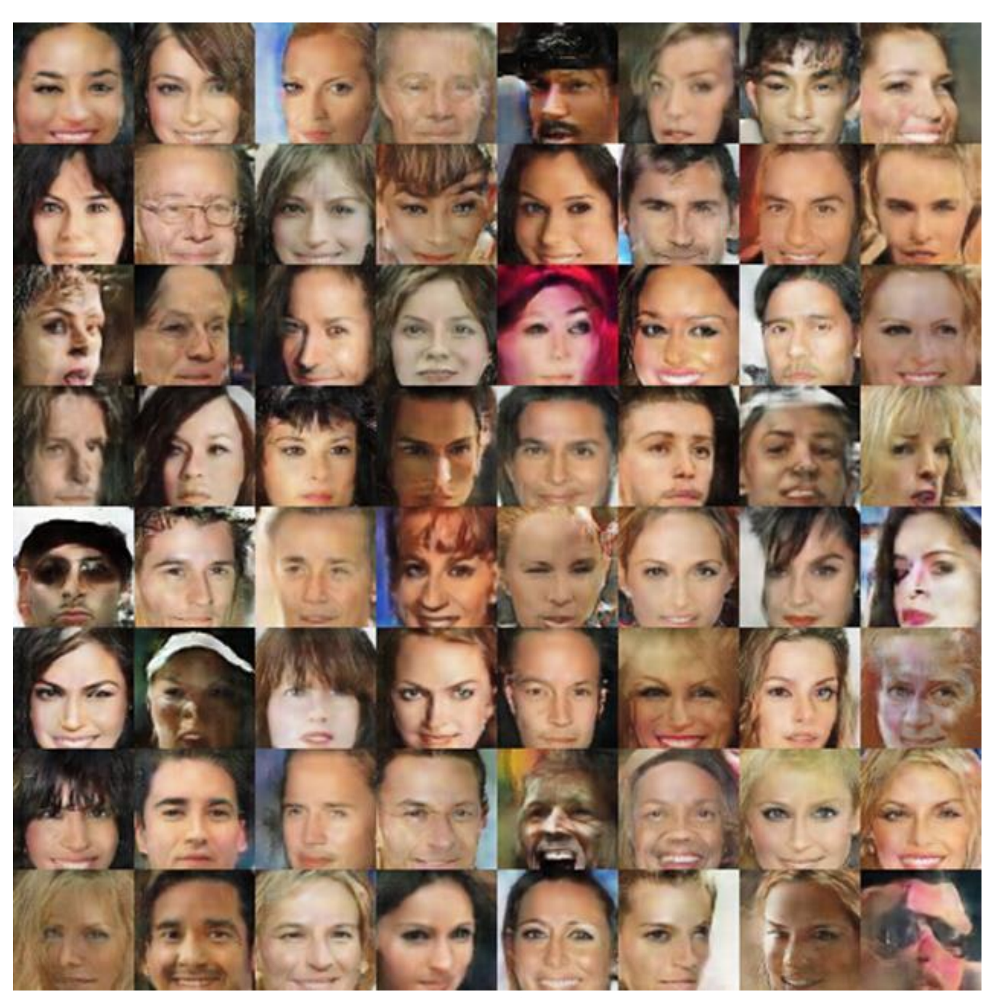
شبكة الخصومة التوليدية

ما هو GAN؟

GAN عبارة عن معمارية شبكة عصبية عميقة تتكون من شبكتين، شبكة المولد generator network وشبكة المميز discriminator network. ومن خلال دورات متعددة من التوليد generation والمميز discrimination، تقوم كلتا الشبكتين بتدريب بعضهما البعض، بينما تحاول كل منهمافي الوقت نفسه التفوق على الأخرى.

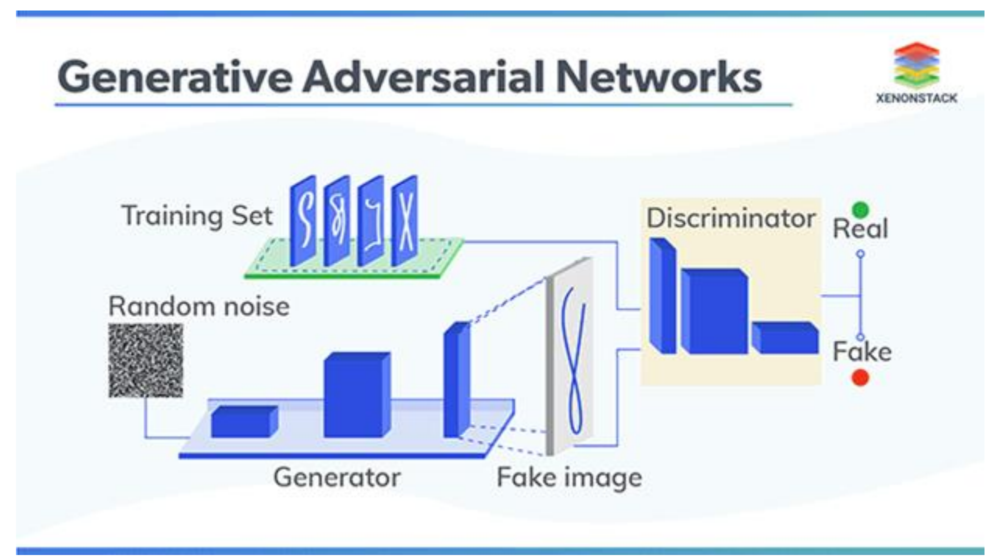
هدفهم هو إنشاء نقاط بيانات تشبه بطريقة سحرية بعض نقاط البيانات الموجودةفي مجموعة التدريب.

حاليًا، يستخدم الأشخاص شبكة GAN لإنشاء ميزات متنوعة. يمكنه إنشاء صور واقعية ونماذج ثلاثية الأبعاد ومقاطع فيديو وغير ذلك الكثير.



توليد الوجوه باستخدام DCGAN

أولاً، دعونا نلقي نظرة على نموذج شبكات GAN العامة.



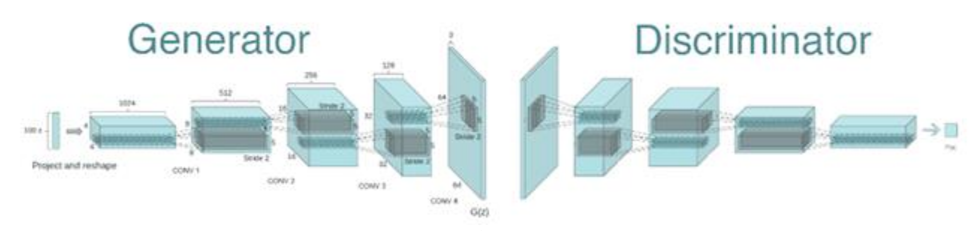
معمارية الشبكات الخصومة التوليدية

ما هي شبكة المولد؟

تستخدم شبكة المولد البيانات الموجودة لإنشاء بيانات جديدة. ويمكنه، على سبيل المثال، استخدام الصور الموجودة لإنشاء صور جديدة. الهدف الأساسي للمولد هو توليد البيانات (مثل الصور أو الفيديو أو الصوت أو النص) من متجه أرقام تم إنشاؤه عشوائيًا، يسمى المساحة الكامنة latent space. أثناء إنشاء شبكة المولدات، نحتاج إلى تحديد هدف الشبكة. قد يكون هذا بمثابة إنشاء الصور، أو إنشاء النص، أو إنشاء الصوت، أو إنشاء الفيديو، وما إلى ذلك.

ما هي شبكة المميز؟

تحاول شبكة المميز التمييز بين البيانات الحقيقية والبيانات التي تولدها شبكة المولد. تحاول شبكة المميز وضع البيانات الواردةفي فئات محددة مسبقًا. يمكنه إما إجراء تصنيف متعدد الفئات أو تصنيف ثنائي. بشكل عام، يتم إجراء التصنيف الثنائيفي شبكات GAN.

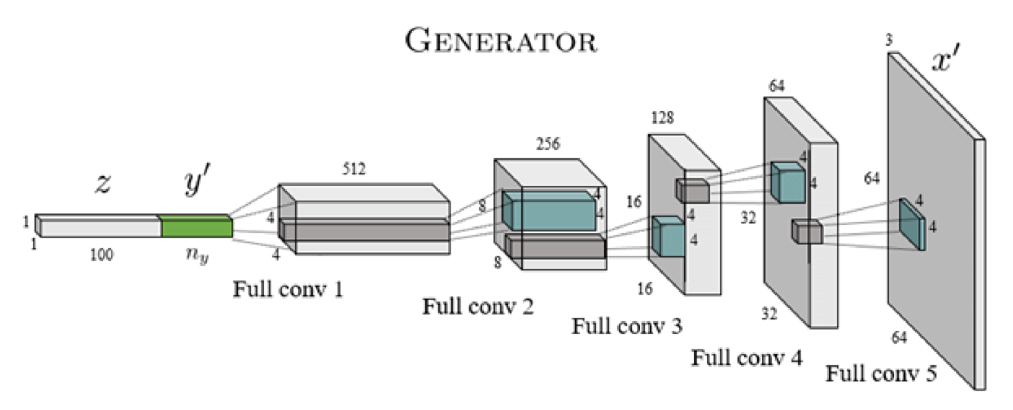


شبكة المولد والمميزفي شبكات GAN

التدريب من خلال اللعب التنافسي في شبكات GAN

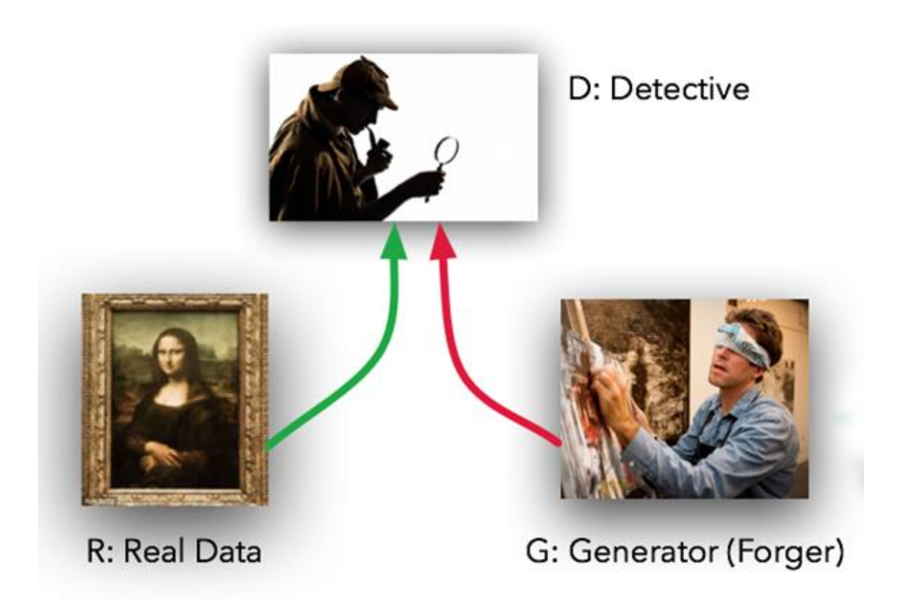
في شبكة GAN، يتم تدريب الشبكات من خلال اللعب التنافسي adversarial play: حيث تتنافس الشبكتان ضد بعضهما البعض. على سبيل المثال، لنفترض أننا نريد من GAN إنشاء أعمال فنية مزيفة:

- الشبكة الأولى، المولد generator، لم يسبق لها رؤية العمل الفني الحقيقي ولكنها تحاول إنشاء عمل فني يشبه الشيء الحقيقي.



تدريب المولد

- أما الشبكة الثانية، وهي شبكة المميز discriminator، فتحاول تحديد ما إذا كان العمل الفني حقيقيًا أم مزيفًا.

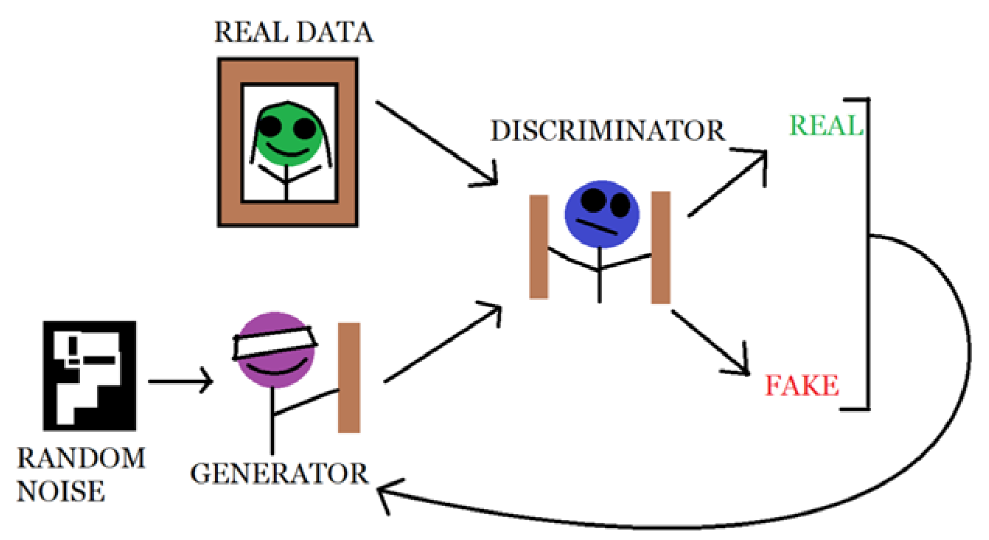


يحاول المولد بدوره خداع المميز ليعتقد أن المنتجات المقلدة هي الصفقة الحقيقية من خلال إنشاء عمل فني أكثر واقعية عبر تكرارات متعددة.

يحاول المُميِّز التفوق على المولد من خلال الاستمرارفي تحسين معاييره الخاصة لتحديد المنتجات المزيفة.

إنهم يرشدون بعضهم البعض من خلال تقديم استجابة من التغييرات الناجحة التي يقومون بهافي عمليتهم الخاصةفي كل تكرار.

في نهاية المطاف، يقوم المُميِّز بتدريب المولد إلى النقطة التي لا يستطيع عندها تحديد العمل الفني الحقيقي وأي العمل الفني المزيف.



كيفية تنفيذ شبكات GAN في مواجهة مشكلة الشيخوخة

هذه تعليمات حول كيفية تنفيذ شيخوخة الوجه باستخدام GAN. يعد تنفيذ شبكات GAN أمرًا صعبًا بعض الشيء.

يتم تنفيذ جميع الكوداتفي TensorFlow 1.12 و9.0 CuDA. نصحك بالتشغيلفي بيئة Python.

تثبيت Cuda 9.0 (قد يستغرق ذلك بضع دقائق)

```
$wget
https://developer.nvidia.com/compute/cuda/9.0/Prod/local_ins
tallers/cuda-repo-ubuntu1604-9-0-local_9.0.176-1_amd64-
deb$dpkg -i cuda-repo-ubuntu1604-9-0-local_9.0.176-1_amd64-
deb$apt-key add /var/cuda-repo-9-0-local/7fa2af80.pub$apt-
get update$apt-get install cuda=9.0.176-1
```

لتثبيت TensorFlow، قم بتشغيل الأمر أدناه في التيرمينال:

```
pip install --upgrade tensorflow-gpu==1.12.2
```

استنساخ هذا الريبو:

```
git clone https://github.com/pbaylies/stylegan-encoder\cd
stylegan-encoder
```

إعداد هيكل المجلد لصورنا:

```
rm -rf aligned_images raw_imagesmkdir aligned_images
raw_images
```

تحضير الصور للتدريب

ضع صورك التي ترغب في تغييرها في المجلد Raw_images، وستكون هيكل البيانات على النحو التالي:

```
├── ./raw_images
│ ├── [your images shoule be here]
│ ├── [your images should be here]
```

محاذاة الوجوه تلقائيًا

تشغيل السكريبت:

```
python align_images.py raw_images/ aligned_images/ --
output_size=1024
```

هذا السكريبت سوف:

1. ابحث عن الوجوه في الصور.
2. قص الوجوه من الصور.
3. قم بمحاذاة الوجوه.
4. قم بإعادة قياس الصور الناتجة وحفظها في مجلد "Aligned_images".

ترميز الوجوه في المساحة الكامنة لـ StyleGAN.

```
$gdown https://drive.google.com/uc?id=1aT59NFy9-
bNyXjDuZOTMl0qX0jmZc6Zb$mkdir data$mv finetuned_resnet.h5
data$rm -rf generated_images latent_representations
```

تدريب الترميز الكامن

```
$python encode_images.py --optimizer=adam --lr=0.002 --
decay_rate=0.95 --decay_steps=6 --use_l1_penalty=0.3 --
face_mask=True --iterations=500 --early_stopping=False --
early_stopping_threshold=0.05 --average_best_loss=0.5 --
use_lpips_loss=0 --use_discriminator_loss=0 --
output_video=True aligned_images/ generated_images/
latent_representations/
```

قم بالوصول إلى https://drive.google.com/drive/u/1/folders/1exoCSLE-CRmfr9yqW3Yv4M9YI7VAw1LZ وقم بتنزيل هذه الملفات المدربة مسبقًا:

ضع هذه الملفات في نفس المجلد.

احفظ outout_vectors.npy في latent_representations بواسطة السكريبت.

```
$python save_latent.py
```

قم بتحرير الملف save_latent.py لتحديد معلمة out_file للحصول على الوجهة الكامنة.

تنفيذ التقدم في شيخوخة الوجه

في المجلد الجذر، قم بتنفيذ:

```
$git clone https://github.com/tr1pzz/InterFaceGAN.git$cd
InterFaceGAN/$gdown
https://drive.google.com/uc?id=1MEGjdvVpUsu1jB4zrXZN7Y4kBBOz
izDQ$mv karras2019stylegan-ffhq-1024x1024.pkl
InterFaceGAN/models/pretrain/karras2019stylegan-ffhq-
1024x1024.pkl
```

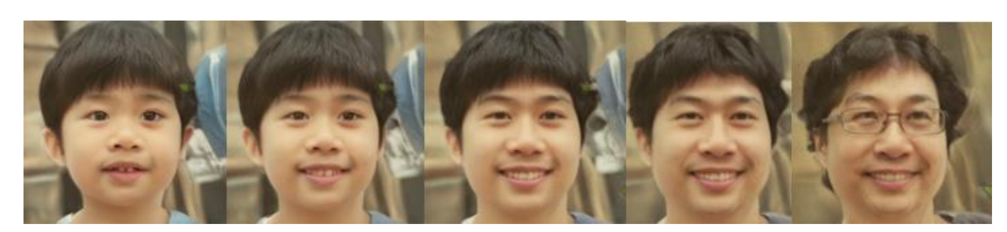
الاختبار

قم بتحميل input_vector.npy إلى Final_w_vectors.

قم بتشغيل هذا الأمر لاستخدام Final_w_vectors لإنشاء الصور.

```
python test_age.py
```

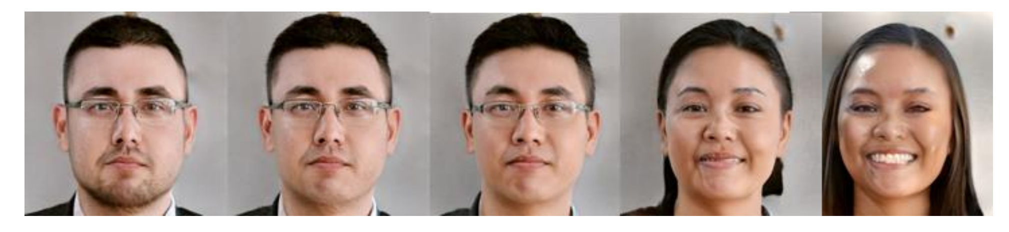
النتائج على النحو التالي:



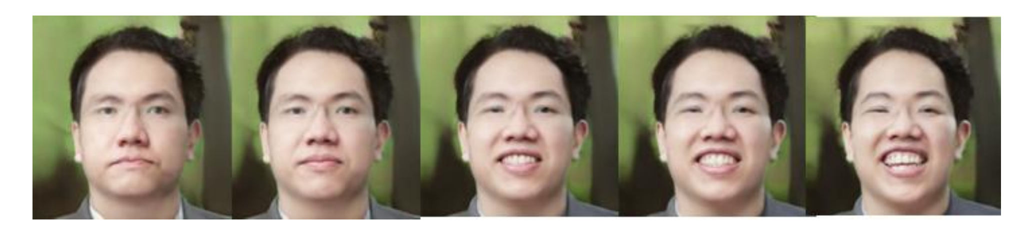
التغيرات في العمر



التغيرات في الجنس



التغيرات في الجنس



التغيرات في الابتسام

المصدر:

https://www.neurond.com/blog/gans-face-aging-problems-try-it-with-your-face

12) نقل النمط باستخدام شبكات الخصومة التوليدية Style Transfer with GANs

مقدمة

استكشفت عدد من الدراسات الحديثة بعض الطرق والتقنيات لإنشاء صور عالية الوضوح (1024 × 1024 بكسل) باستخدام شبكات GAN (شبكات الخصومة التوليدية Generative Adversarial Networks). من المثير للدهشة بشكل لا يصدق رؤية صور عالية الدقة وواقعية للغاية لوجوه بشرية وحيوانات وأشياء أخرى تم إنشاؤها بواسطة خوارزمية، خاصة تذكر صور GAN الأولى منذ بضع سنوات فقط. لقد انتقلنا من الصور ذات الجودة المنخفضة والمنقطة إلى الصور القريبة من الواقع في وقت قصير جدًا: وهذا دليل واضح حقًا على مدى سرعة تقدم الأبحاث في هذا المجال.



لكن عند قراءة هذه الدراسات الحديثة (والأكثر صلة بالموضوع هو بحث StyleGAN الذي أعدته Nvidia وبحث BigGAN الذي أعدته Google) أجد دائمًا الجانب الذي ينجح في تقليل إحساسي بالمفاجأة والإثارة: قوة الحوسبة computing power. إن اكتشاف القدرات الحاسوبية الضخمة المستخدمة لالتقاط تلك الصور يجعلني أدرك أن بيني وبين تلك النتائج عقبة لا يمكن التغلب عليها. هذه الفكرة وحدها تجعلني أشعر أن التكنولوجيا الجديدة برمتها التي تم استكشافها في الدراسات تبدو لي بعيدة جدًا، وبالتالي أقل إثارة للدهشة.

ولهذا السبب، أود في هذه المقالة استكشاف كيفية جعل شبكات GAN والصور عالية الدقة تعمل معًا دون الحاجة إلى أجهزة باهظة الثمن، مما يفتح فرصًا جديدة للأشخاص الذين ليس لديهم بالضرورة إمكانية الوصول إلى وحدات معالجة الرسوميات GPU عالية المستوى. يمكن تحقيق كل ما تم شرحه هنا باستخدام منصة Google Colaboratory المتاحة مجانًا، والتي توفر وحدة معالجة رسوميات GPU مجانية لجميع مشاريع التعلم الآلي/العميق الخاصة بك.

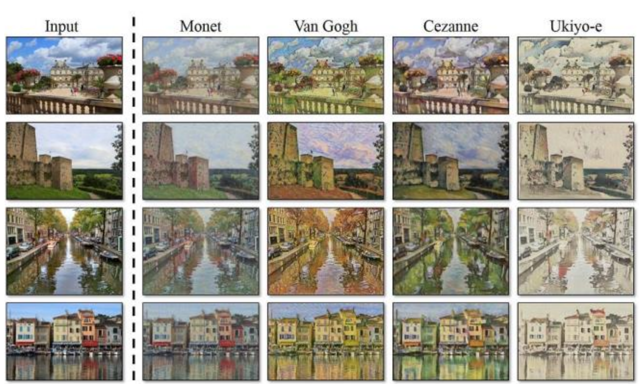
هدفنا

سنحاول إجراء نقل النمط Style Transfer بين مجالين من الصور عالية الوضوح، باستخدام معمارية GAN خاصة ولكن بسيطة لأداء مهمتنا. وبشكل أكثر تحديدًا، سنقوم بتطبيق نمط رسم painting style فان جوخ Van Gogh على صور عالية الدقة للمناظر الطبيعية. من العدل أن نقول إن نقل النمط كان موضوعًا عصريًافي الرؤية الحاسوبية Computer Vision خلال السنوات القليلة الماضية؛ الورقة الأصلية التي بدأت هذا الاتجاه هي "الخوارزمية العصبية للأسلوب الفني" (جاتيس وآخرون A Neural Algorithm of Artistic Style" (Gatys et al.)، والتي استخدمت خطأ المحتوى والنمط Content and Style loss على شبكة تلافيفية مُدربة مسبقًا pretrained convolutional network لأداء المهمة. على الرغم من أن هذه الطريقة يمكن أن تعمل على الصور عالية الدقة، إلا أنها يمكنها فقط استخدام صورة واحدة (دعنا نقول "Starry Night") كتمثيل لأسلوب الرسام، وهو ما ليس ما نريده.



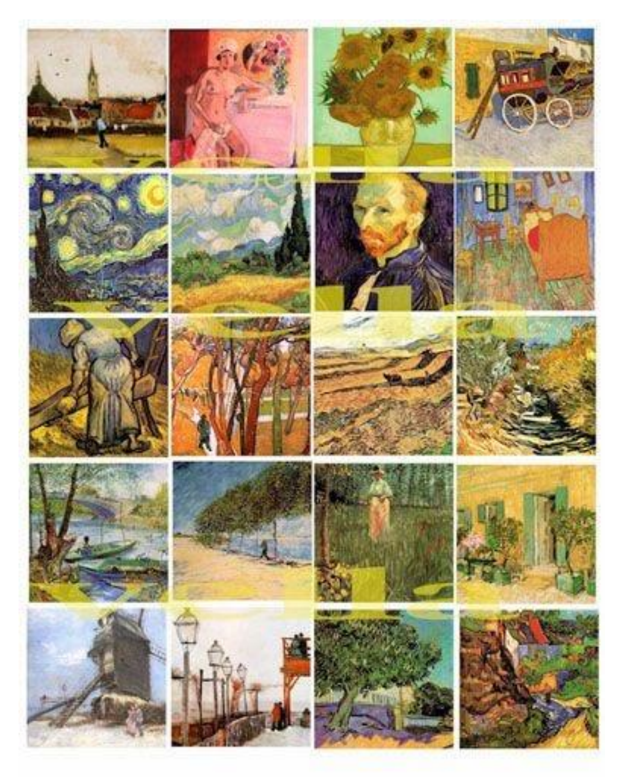
أمثلة على نقل النمط من الورقة الأصلية

من ناحية أخرى، تحتاج GAN بشكل عام إلى مجال من الصور للتدريب عليه، وبالتالي فهي قادرةفي حالتنا على التقاط أسلوب الرسام بالكامل (يظهر بحث CycleGAN نتائج مثيرة للاهتمام حول نقل النمط).



أمثلة على نقل نمط CycleGAN

ومع ذلك، فإن تدريب شبكات GAN يعد مكلفًا للغاية من الناحية الحسابية: حيث لا يمكن إنشاء صور عالية الدقة إلا باستخدام أجهزة متطورة للغاية وأوقات تدريب طويلة. آمل أن تكون الحيل والتقنيات الموضحةفي هذه المقالة قادرة على مساعدتكفي مغامرات توليد الصور عالية الدقة.



لوحات فان جوخ



صور عالية الدقة مترجمة إلى نمط فان جوخ

هيا نبدأ!

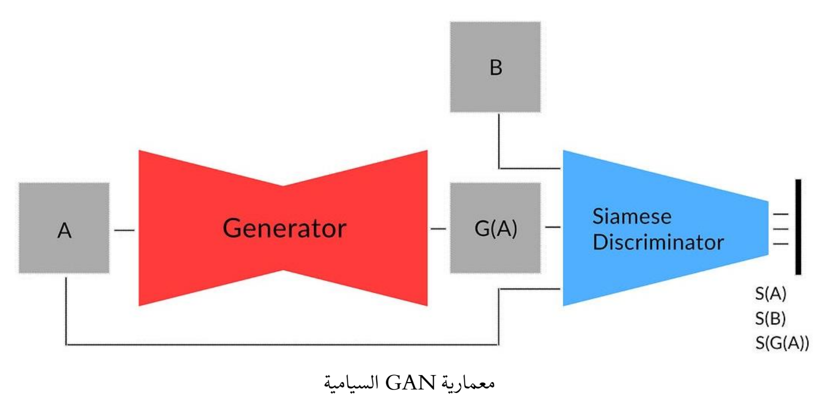
المعمارية

ما سنحاول تحقيقه يسمى ترجمة صورة إلى صورة image-to-image translation (من المجال A إلى المجال B). هناك طرق مختلفة ومعماريات شبكية لتحقيق ذلك: قد يكون أشهرها CycleGAN، ولكن يوجد أيضًا عدد من الأوراق البحثية الأخرى حول نفس الموضوع.

في تجاربي، استخدمت معمارية مخصصة تتكون من شبكة سيامية Siamese Network كمميز Discriminator ودالة خطأ loss function خاصة (ولكنها سهلة للغاية). لقد اخترت هذه الطريقة لأنها لا تعتمد على الاختلافات لكل بكسل في أي من الاخطاء: وهذا يعني ببساطة أن الشبكة لا تتبع أي قيود هندسية على الصورة التي تم إنشاؤها وبالتالي فهي قادرة على إنشاء ترجمات أكثر إقناعا للصور (هذا صحيح في حالتنا).

يمكن العثور على شرح عميق وشامل لهذا النوع من المعمارية وكيفية عمله في هذه المقالة الأخرى التي كتبتها هنا.

فيما يلي مقدمة مختصرة عن معمارية GAN السيامية (Siamese GAN).



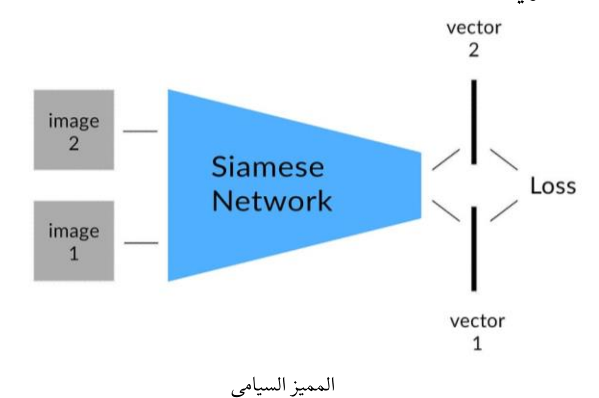
معمارية GAN السيامية

وهي مصنوعة من مولد واحد (G) ومميز (D): يأخذ G الصورة كمدخل ويخرج الصورة المترجمة translated image؛ يأخذ D صورة كمدخل ويخرج متجهاً كامناً latent vector.

لدى المميز السيامي Siamese Discriminator هدفين: إخبار G بكيفية إنشاء صور أكثر واقعية والحفاظ على الارتباط في تلك الصور المزيفة (نفس "المحتوى content") مع الصور الأصلية.

باستدعاء الصور العشوائية A1 وA2 وB1 وB2 من المجالات A وB على التوالي، وX صورة عشوائية، وG(X) الصور التي تم إنشاؤها بواسطة المولد، يجب على المميز ترميز الصور إلى متجهات D(X) مثل:

1. يجب أن تكون D(B1) قريبة (المسافة الإقليدية Euclidean distance) من نقطة ثابتة (نقطة الأصل على سبيل المثال)،في حين يجب أن تكون D(G(A1)) بعيدة عن نفس النقطة. وبالتالي فإن المتجهات الأقرب إلى النقاط الثابتة تمثل صورًا أكثر واقعية. من ناحية أخرى، يحاول المولد تقليل المسافة من D(G(A1)) إلى النقطة الثابتة، بطريقة عدائية كلاسيكية.
2. (D(A1)-D(A2)) يجب أن يكون مشابهًا (تشابه جيب التمام cosine similarity) مع (D(G(A1))-D(G(A2)))، للحفاظ على "المحتوى" بين A و G(A)). يشارك كل من المولد والمميزفي هذا الهدف.



المميز السيامي

مع وجود هذين القيدين (الاخطاء)، الأول يعتمد على حجم المتجهات بينما يعتمد الثاني على الزاوية بين المتجهات، يتم تحقيق هدفنا الكامل ويمكننا تحقيق هدفنا النهائي المتمثل في ترجمة الصورة إلى الصورة من المجال A إلى المجال B. أقترح عليك حقًا قراءة هذا المقال حيث أقدم شرحًا أكثر شمولاً وعمقًا لهذه المعمارية، مع عرض الرسوم التوضيحية والأمثلة.

الآن بعد أن أصبح لدينا المعمارية مقفلة، فلنستكشف كيف وماذا يجب تغذية الشبكة من أجل الوصول إلى إنشاء صور عالية الدقة.

استخراج الصور

نحتاج إلى مجموعتي بيانات من الصور عالية الوضوح:في حالتنا سنستخدم مجموعة بيانات من المناظر الطبيعية (المجال A) ومجموعة بيانات من لوحات فان جوخ (المجال B). ضعفي اعتبارك أنه كلما كانت الصور التي تختار العمل بها أكبر، كلما استغرقت المعالجة المسبقة (قصها وتغيير حجمها) تلك الصور وقتًا أطول (على الرغم من أن ذلك لن يزيد من الوقت الذي تقضيه حصريًافي تدريب الشبكة!).

نحن الآن بحاجة إلى اختيار حجم الصور التي سيتم تغذيتها إلى المولد: من الواضح أننا لا نستطيع استخدام حجم الصور عالية الدقة بأكملها من مجموعات البيانات، وإلا فإن أوقات التدريب وأحجام

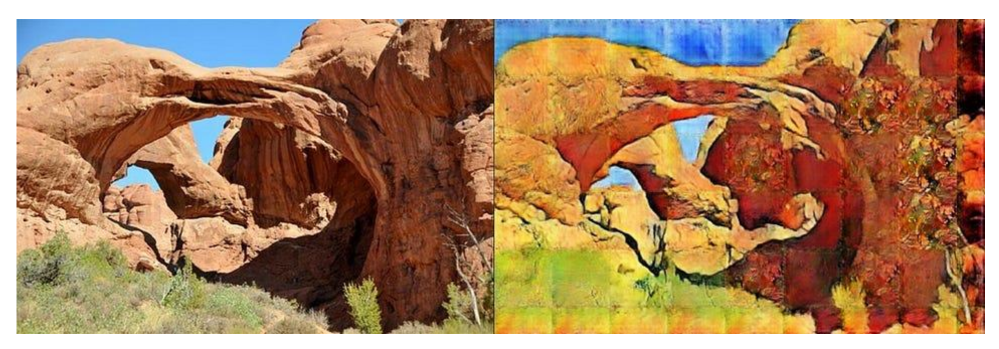
الشبكة ستكون ضخمة ولن يتم حل أي مشكلة. وبالتالي، نختار حجمًا صغيرًا بدرجة كافية SxS (64 × 64 بكسل على سبيل المثال) بحيث يمكن التحكم في أوقات التدريب ويظل كل شيء ممكنًا من الناحية الحسابية حتى بالنسبة لوحدات معالجة الرسوميات المتوسطة (مثل تلك المتاحة مجانًا على Google Colaboratory).

وبالتالي، كما كنت تعتقد، فإن الصور، قبل أن يتم تغذيتها إلى المولد، يجب أن يتم قصها (أو اقتصاصها) إلى صور SxS أصغر. وهكذا، بعد قراءة الصورة وتحويلها إلى موتر، نقوم بإجراء قص SxS عشوائي على الصورة، وإضافتها إلى دفعة وتغذية الدفعة إلى الشبكة. يبدو الأمر سهلاً للغاية وهو كذلك بالفعل!

الآن، لنفترض أننا قمنا بتدريب GAN باستخدام هذه الطريقة حتى تتم ترجمة كل محصول SxS صغير إلى أسلوب Van Gogh بواسطة المولد بطريقة تُرضينا: كيف يمكننا الآن ترجمة صورة عالية الدقة بالكامل من المجال A إلى المجال B؟

مرة أخرى، الأمر بسيط للغاية: يتم تقسيم الصورة إلى أجزاء SxS صغيرة (إذا كان حجم الصورة عالية الدقة هو BxB، فسيكون لدينا (B//S)x(B//S) صور SxS صغيرة)، تتم ترجمة كل صورة SxS بواسطة المولد، وأخيرًا تم ضم كل شيء معًا مرة أخرى.

ومع ذلك، إذا حاولنا تدريب GAN باستخدام هذه الفكرة البسيطة المتمثلة في استخراج الصور الأصغر من الصور الأكبر حجمًا، أثناء وقت الاختبار، فسنلاحظ قريبًا مشكلة مزعجة تمامًا: الصور الصغيرة المستخرجة بواسطة الصورة الكبيرة التي نريد ترجمتها، عند تحويلها بواسطة المولد إلى المجال B، لا تمتزج عضويًا معًا do not blend organically together. تظهر حواف كل صورة SxS بوضوح في التركيبة النهائية، مما يفسد "سحر" عملية نقل النمط الناجحة. هذه مشكلة صغيرة نسبيًا ويمكن أن تكون مزعجة للغاية: حتى باستخدام الأساليب المعتمدة على البكسل مثل CycleGAN، لا تزال نفس العقبة تظهر.



الحواف مرئية

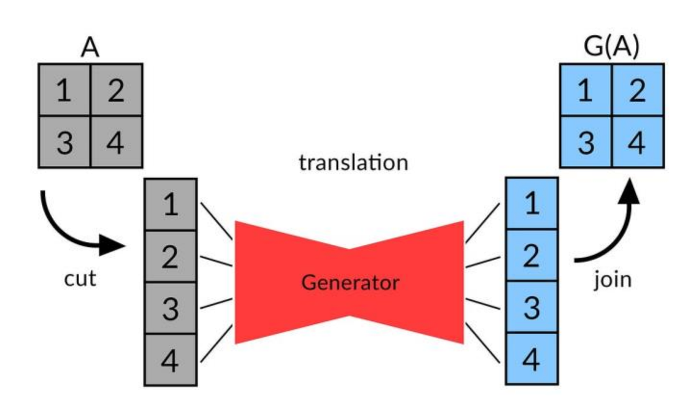
كيف يمكننا حل هذه المشكلة؟

الحل الذي استخدمته سهل الفهم وأنيق للغاية في رأيي، وهو يمثل الفكرة الأساسية التي أتمنى أن تتذكرها (وربما تستخدمها) من هذه المقالة.

أولاً، نحتاج إلى إعادة النظر في خط سير البيانات الخاص بنا: بينما قبل أن نقطع اقتصاصات SxS مباشرةً من صورة BxB HD، يتعين علينا الآن الحصول على اقتصاصات **2Sx2S** (إذا كانت S = 64، فإننا نحتاج إلى اقتصاصات 128x128). بعد ذلك، بعد تعريف المولد الخاص بنا، نقوم بإنشاء نموذج جديد، يسمى Combo، يقوم بالعمليات التالية:

1. خذ مجموعة من الصور **2Sx2S** (من المجال A) كمدخل (INP)؛
2. قص كل صورة في INP إلى 4 صور SxS (INPCUT)؛
3. قم بتغذية كل صورة من صور INPCUT الأربع SxS إلى المولد واحصل على OUTCUT (نفس الشكل الدقيق لـ INPCUT، ولكن مع نسخة مترجمة من كل صورة SxS)؛
4. انضم إلى كل مجموعة مكونة من 4 صور SxS في OUTCUT واخرج (نفس الشكل الدقيق لـ INP، ولكن مع نسخة مترجمة من كل صورة 2Sx2S)؛

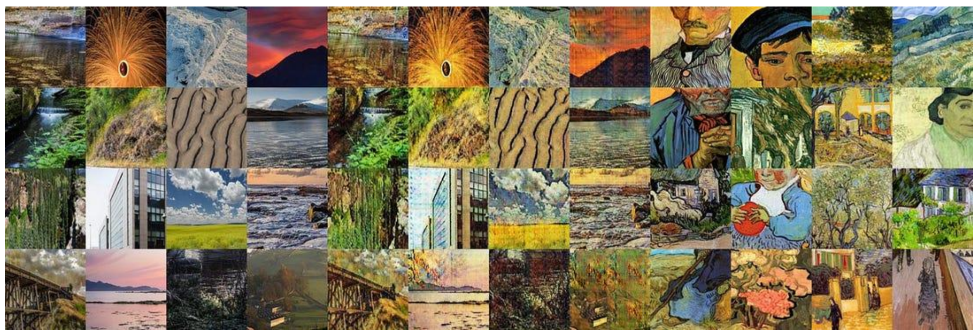
5. الإخراج.



نموذج Combo: الاقتصاص والترجمة والانضمام.

يتم بعد ذلك تمرير مخرجات Combo كمدخل إلى المميز، الذي يقبل الآن مدخلات ذات حجم مضاعف عما كان عليه سابقاً (2Sx2S). لا يتطلب هذا التعديل الصغير وقتاً أطول بكثير للحساب ويمكنه حل مشكلتنا السابقة بشكل فعال. كيف؟

يضطر المولد الآن إلى إنشاء صور متماسكة فيما يتعلق بالحواف والألوان، لأن المميز لن يصنف الصور المرتبطة غير المتماسكة على أنها واقعية وبالتالي سيخطر المولد بالمكان الذي يمكنه تحسينه. بالغوص بشكل أعمق قليلاً، يضطر المولد إلى تعلم كيفية إنشاء حواف واقعية على كل من الحواف الأربعة لصورة SxS:في الصورة النهائية 2x2المرتبطة، تكون كل حافة من الحواف الأربعة على اتصال مع حافة أخرى، وحتى حافة واحدة تم إنشاؤها بشكل سيئ من شأنها أن تدمر واقعية الصور 2x2.



العينات أثناء التدريب: (من اليسار إلى اليمين) صور من المجال A، صور مترجمة (AB)، صور من المجال B

كل الأشياء معًا

للتأكد من أن كل شيء حتى هنا واضح ومفهوم، دعونا نلخص كيفية عمل الشبكة بأكملها.

الهدف هو تطبيق نمط B على الصور الموجودةفي A. يتم قطع الصور ذات الحجم 2Sx2S من صور عالية الدقةفي كلا النطاقين A وB. الصور من A هي مدخلات Combo؛ يقوم هذا النموذج بتقطيع الصور إلى 4 صور أصغر (SxS)، ثم يستخدم المولد G لتحويلها، وفي النهاية يجمعها معًا. نحن نسمي هذه الصور المزيفة AB.

الآن دعونا نركز على المميز السيامي D: حجم مدخلاته هو ضعف حجم مدخلات المولد (2Sx2S)، في حين أن الإخراج هو متجه بحجم LENVEC.

يقوم D بترميز الصور إلى متجهات D(X) مثل:

1. يجب أن يكون D(B) قريبًا من الأصل (متجه الأصفار بحجم VECLEN):

LossD1 هو مربع المسافة الإقليدية لـ D(B) من نقطة الأصل، لذا $Eucl(D(A))^2$;

2. يجب أن يكون D(AB) بعيداً عن نقطة الأصل:

LossD2 هي $(max(0, cost - Eucl(D(AB))))^2$

3. يجب أن تكون متجهات التحويل ((D(AB1)-D(AB2)) and (D(A1)-D(A2)) متجهات متشابهة، للحفاظ على "محتوى" الصور:

LossD3 هو **cosine_similarity(D(A1)-D(A2), D(AB1)-D(AB2))**

من ناحية أخرى، يجب على المولد إنشاء صور (منضمة joined) AB مثل:

5. يجب أن يكون D(AB) قريبًا من نقطة الأصل:

LossG1 هو $Eucl(D(AB))^2$

6. يجب أن تكون متجهات التحويل ((D(AB1)-D(AB2)) and (D(A1)-D(A2)) متجهات متشابهة (نفس هدف المميز):

LossG2 هو **cosine_similarity(D(A1)-D(A2), D(AB1)-D(AB2))**

يمكن العثور على شرح أعمق حول كيفية عمل كل واحدة من هذه الاخطاءفي مقالتي هنا، حيث أشرح بالتفصيل كيفية عمل المميز السيامي (أعتقد أنها تستحق القراءة!).

هذا كل شيء!

باتباع هذه الطريقة، يستطيع المولد تعلم كيفية إنشاء صور صغيرة منمقة يمكن ضمها معًا دون أي تناقض في الحواف. وبالتالي، عند ترجمة صورة كاملة عالية الدقة، بعد تقطيعها إلى صور SxS منفصلة أصغر حجمًا وإدخالها إلى المولد، يمكننا دمجها معًافي صورة عالية الدقة نهائية وممتعة بصريًا ومتماسكة.





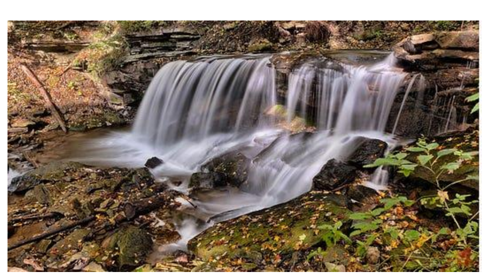


أمثلة الترجمة على الصور عالية الدقة: على الرغم من أنها ليست مثالية، إلا أنه يتم إنشاء ضربات فرشاة واقعية وتبدو الصور متماسكة تمامًا.
قد تكون الحلول هي ضبط الشبكات وسعة أكبر

الاستنتاج

لا تزال التقنية الموضحةفي هذه المقالة تعرض بعض المشكلات التي نحتاج إلى معالجتها.

إذا تم اختيار صور عالية الوضوح للغاية، فإن الاشياء الصغيرة المستخدمة لتدريب الشبكة قد لا تحتوي على أي معلومات ذات صلة (قد تكون مجرد ألوان صلبة، تشبه وحدات البكسل المفردة) وبالتالي قد لا يكون التدريب ناجحًا: يحتاج كل من المولد والمميز نوع ما من المعلومات المراد معالجتها (يجب على المميز تشفير الصور بناءً على "محتواها") وقد يواجه بعض المشكلات إذا لم تكن هذه المعلومات متاحة.




حالة الفشل: "يهلوس" المولد بألوان وأشكال غير متماسكةفي بعض المناطق

حتى لو انتهى التدريب بنجاح، عند ضم جميع الاقتصاصات المختلفة لصورة ذات دقة عالية جداً، فإن المساهمة الأسلوبية لكل صورة صغيرة مترجمة ليست كافية لكامل الصورة عالية الدقة، والتي غالبًا ما تبدو مشابهة للصورة الأصلية مع تغيير فقطفي ألوانها.

في تجاربي، وجدت أنه بالنسبة لمرحلة التدريب، فإن استخدام نسخة تم تغيير حجمها (دقة أقل) من مجموعة البيانات عالية الدقة، أثناء التبديل إلى الصور عالية الدقة بأكملها عند الترجمة، يساعد بالتأكيد فيما يتعلق بالمشكلة الأولى.

تترك هذه التقنية الكثير مما يجب استكشافه: يمكن أن يكون من الممكن إجراء أنواع أخرى من ترجمات الصور تختلف عن نقل النمط التقليدي. من المهم أن تتذكر أن المولدفي الحالة المعروضة ليس لديه أي فكرة عن سياق الصورة عالية الدقة بأكملها و"يرى" فقط الاشياء ذات الدقة المنخفضة. وبالتالي، فإن إعطاء المولد بعض السياق (ربمافي شكل "متجه سياق context vector" مشفر؟) حول الصور بأكملها يمكن بالتأكيد أن يوسع نطاق تطبيقات هذه التقنية، مما يفتح إمكانيات لأنواع أكثر تعقيداً "مدركة للسياق context aware" من ترجمات الصور عالية الدقة (الأشياء إلى الأشياء الأخرى والوجوه والحيوانات).

لذا، كما كنت قد فهمت، فإن الاحتمالات لا حصر لها ولم يتم اكتشافها بعد!

المصدر:

https://towardsdatascience.com/style-transfer-with-gans-on-hd-images-88e8efcf3716

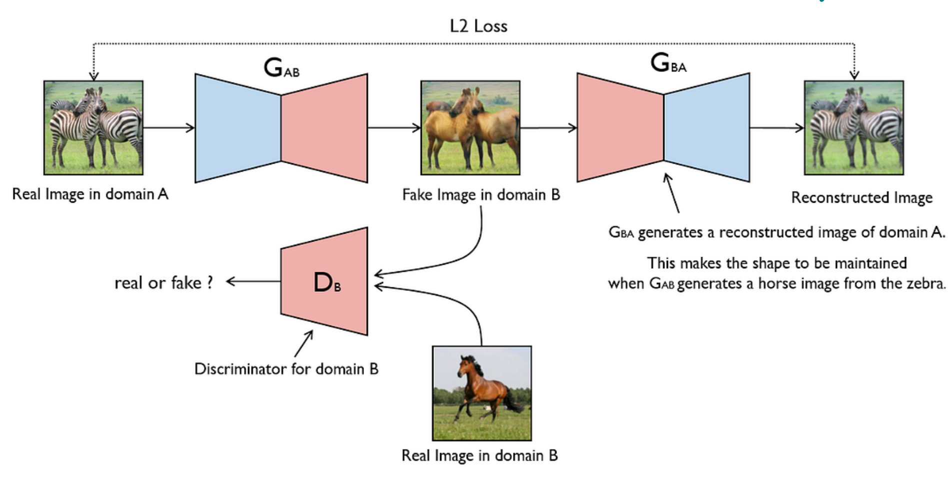
13) ترجمة الصورة باستخدام شبكات الخصومة التوليدية

Image Translation using GANs

شبكة الخصومة التوليدية للدورة Generative Adversarial Network(CycleGAN)، هي طريقة لتدريب الشبكات التلافيفية العميقة deep convolutional networks لمهام ترجمة الصورة إلى الصورة Image-to-Image translation. على عكس نماذج GAN الأخرى لمهام ترجمة الصور، تتعلم CycleGAN التعيين mapping بين مجال صورة وآخر باستخدام نهج غير خاضع للأشراف. على سبيل المثال، إذا كنا مهتمين بترجمة صورة حصان إلى صورة حمار وحشي، فإننا لا نطلب تحويل مجموعة بيانات التدريب الخاصة بالحصان فعليًا إلى حمار وحشي. الطريقة التي تقوم بها CycleGAN بذلك هي من خلال تدريب شبكة المميز Generator Networks على تعلم التعيين من المجال X (domain X) إلى صورة تبدو وكأنها جاءت من المجال Y (domain Y)(والعكس صحيح).

سوف تحصل على فهم أعمق لكيفية القيام بذلك أثناء سيرك في هذه المقالة. لذلك دعونا نبدأ...

CycleGAN



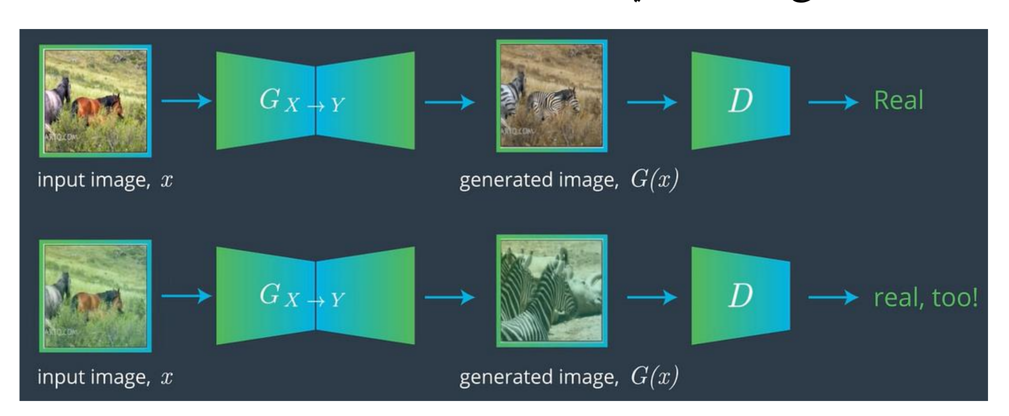
معمارية CycleGAN

بالنسبة لمجموعة الصور المقترنة paired images، يمكننا إنشاء GAN مباشرة لتعلم التعيين من x إلى y بمساعدة Pix2Pix. يمكنك قراءة المزيد عن شبكات Pix2Pix هنا.

لكن إعداد مجموعات مقترنة من البيانات يستغرق وقتًا طويلاً وصعبًا. على سبيل المثال، ما أعنيه بالمجموعة المقترنة هو أننا بحاجة إلى صورة حمار وحشيفي نفس وضع الحصان أو بنفس الخلفية حتى نتمكن من تعلم التعيين.

لتكون قادرة على حل هذه المشكلة، تم تطوير معمارية CycleGAN. تمكن CycleGANs من تعلم التعيين من مجال X إلى مجال آخر Y دون الحاجة إلى العثور على أزواج تدريب متطابقة تمامًا! دعونا نلقي نظرة على كيفية قيام CycleGAN بذلك.

لنفترض أن لدينا مجموعة من الصور من المجال X ومجموعة غير مقترنة unpaired images من الصور من المجال Y. نريد أن نكون قادرين على ترجمة صورة من مجموعة إلى أخرى. للقيام بذلك، نحدد تعيين G(G: X->Y) الذي يبذل قصارى جهده لتعيين X إلى Y. ولكن مع البيانات غير المقترنة، لم تعد لدينا القدرة على النظر إلى أزواج البيانات الحقيقية والمزيفة. لكننا نعلم أنه يمكننا تغيير نموذجنا لإنتاج مخرجات تنتمي إلى المجال المستهدف.



لذلك عندما تقوم بدفع صورة حصان (المجال X)، يمكننا تدريب المولد لإنتاج صور واقعية المظهر للحمير الوحشية (المجال Y). لكن المشكلةفي ذلك هي أننا لا نستطيع إجبار مخرجات المولد على التوافق مع مدخلاته (في الصورة أعلاه، التحويل الأول هو الترجمة الصحيحة من صورة إلى صورة). يؤدي هذا إلى مشكلة تسمى انهيار الوضع mode collapse حيث قد يقوم النموذج بتعيين مدخلات متعددة من المجال X إلى نفس الإخراج من المجال Y.في مثل هذه الحالات، بالنظر إلى حصان الإدخال (المجال X)، كل ما نعرفه هو أن الإخراج يجب أن يبدو كما يلي: حمار وحشي (المجال Y). ولكن للحصول على التعيين الصحيح للمدخلاتفي المجال المستهدف المقابل، نقدم تعيينًا إضافيًا مثل التعيين العكسي inverse mapping G'(G': Y->X) الذي يحاول تعيين Y إلى X. وهذا ما يسمى قيد اتساق الدورة cycle-consistency constraint.

فكرفي الأمر على هذا النحو، إذا قمنا بترجمة صورة حصان (المجال X) إلى صورة حمار وحشي (المجال Y)، ثم قمنا بالترجمة مرة أخرى من حمار وحشي (المجال Y) إلى حصان (المجال X)، فإننا يجب أن نعود إلى نفس صورة الحصان التي بدأنا بها.

يجب أن تعيدك دورة الترجمة الكاملة إلى نفس الصورة التي بدأت بها.في حالة ترجمة الصورة من المجال X إلى المجال Y، إذا تحقق الشرط التالي، نقول إن تحويل الصورة من المجال X إلى المجال Y كان صحيحا.

$$G_{YtoX}(G_{XtoY}(x)) \approx x$$

الحالة ـ 1

بمساعدة قيد اتساق الدورة، تتأكد CycleGAN من أن النموذج يتعلم التعيين الصحيح من المجال X إلى المجال Y.

مهمة الترجمة من صورة إلى صورة

يتم تقسيم المهمة التالية إلى سلسلة من المهام الصغيرة بدءًا من تحميل البيانات وتصورها وحتى نماذج التدريب.

تصور مجموعة البيانات

وعلى وجه التحديد، سنلقي نظرة على مجموعة من الصور لمنتزه يوسمايت الوطني Yosemite National Park التي تم التقاطها إما خلال الصيف أو الشتاء. الفصول هي المجالين لدينا!



صور من مجال موسم الصيف.



صور من مجال فصل الشتاء.

بشكل عام، يمكنك أن ترى أن الصور الصيفية أكثر إشراقا وأكثر خضرة من الصور الشتوية. يحتوي الشتاء على أشياء مثل الثلج والصور الغائمة.في مجموعة البيانات هذه، سيكون هدفنا الرئيسي هو تدريب المولد الذي يتعلم تحويل الصورة من الصيف إلى الشتاء والعكس. لا تحتوي هذه الصور على تسميات ويشار إليها ببيانات التدريب غير المقترنة unpaired training data. ولكن باستخدام CycleGAN يمكننا تعلم التعيين من مجال صورة إلى آخر باستخدام النهج غير الخاضع للأشراف.

يمكنك تحميل البيانات التالية بالضغط هنا.

تعريف النماذج

يتكون CycleGAN من مميزين (D_x وD_y) ومولدين (G_xtoy وG_ytox).

- **D_x**: يحدد صور التدريب من المجال X على أنها حقيقية والصور المترجمة من المجال Y إلى المجال X على أنها مزيفة.
- **D_y**: يحدد صور التدريب من المجال X على أنها حقيقية والصور المترجمة من المجال Y إلى المجال X على أنها مزيفة.
- **G_xtoy**: يترجم الصور من المجال X إلى المجال Y.
- **G_ytox**: يترجم الصور من المجال Y إلى المجال X.

المميز

إن المميزات D_x وD_y،في CycleGAN، عبارة عن شبكات عصبية تلافيفية convolutional neural networks ترى الصورة وتحاول تصنيفها على أنها حقيقية أو مزيفة.في هذه الحالة، يُشار إلى الحقيقي بمخرج قريب من 1 ومزيف بالقرب من 0. تتمتع المميزات بالمعمارية التالية:

```
# helper conv function
def conv(in_channels, out_channels, kernel_size, stride=2,
padding=1, batch_norm=True):
"""Creates a convolutional layer, with optional batch
normalization.
"""
layers = []
conv_layer = nn.Conv2d(in_channels=in_channels,
out_channels=out_channels,
kernel_size=kernel_size, stride=stride, padding=padding,
bias=False)

layers.append(conv_layer)

if batch_norm:
layers.append(nn.BatchNorm2d(out_channels))
return nn.Sequential(*layers)
```

```
class Discriminator(nn.Module):

def __init__(self, conv_dim=64):
super(Discriminator, self).__init__()

# Define all convolutional layers
# Should accept an RGB image as input and output a single
value
self.layer_1 = conv(3,conv_dim,4,batch_norm = False)
self.layer_2 = conv(conv_dim,conv_dim*2,4)
self.layer_3 = conv(conv_dim*2,conv_dim*4,4)
self.layer_4 = conv(conv_dim*4,conv_dim*8,4)
self.layer_5 = conv(conv_dim*8,1,4,1,batch_norm = False)

def forward(self, x):
# define feedforward behavior
x = F.relu(self.layer_1(x))
x = F.relu(self.layer_2(x))
x = F.relu(self.layer_3(x))
x = F.relu(self.layer_4(x))

x = self.layer_5(x)
return x
```

الشرح:

- تتكون المعمارية التالية من خمس طبقات تلافيفية convolutional layers تنتج لوجيت logit واحداً. يحدد هذا اللوغاريتم ما إذا كانت الصورة حقيقية أم لا. لا توجد طبقة متصلة بالكامل fully connected layer في هذه المعمارية.
- جميع الطبقات التلافيفية، باستثناء الطبقة الأولى والأخيرة، يتبعها تسوية بالدفعات batch normalization (محددة في دالة مساعد التحويل conv helper function).
- بالنسبة للوحدات المخفية hidden units، يتم استخدام دالة تفعيل ReLU.
- يعتمد عدد خرائط الميزات feature maps بعد كل التفاف على المعلمة conv_dim (في تطبيقي conv_dim = 64).
- يتمتع كل من D_x وD_y بنفس المعمارية، لذلك نحتاج فقط إلى تحديد فئة واحدة، ثم إنشاء مثيلين للمميز لاحقاً.

الكتل المتبقية والدالة المتبقية

أثناء تعريف معمارية المولد، سنستخدم شيئًا يسمى كتلة Resnet (Resnet Block) والدالة المتبقية (residual function) في معماريتنا. الفكرة وراء استخدام كتلة Resnet والدالة المتبقية هي كما يلي:

الكتلة المتبقية

تقوم الكتلة المتبقية بتوصيل المشفر encoder ومفكك الشفرة decoder. الدافع وراء هذه المعمارية هو كما يلي: قد يكون من الصعب جداً تدريب الشبكات العصبية العميقة، حيث من المرجح أن يكون لديها تدرجات gradients متفجرة exploding أو متلاشية vanishing، وبالتالي، تواجه صعوبةفي الوصول إلى التقارب convergence؛ التسوية بالدفعات تساعدفي هذا قليلاً.

أحد الحلول لهذه المشكلة هو استخدام كتل Resnet التي تسمح لنا بتعلم ما يسمى بالدوال المتبقية عند تطبيقها على مدخلات الطبقة.

الدالة المتبقية

عندما نقوم بإنشاء نموذج التعلم العميق، يكون النموذج (عدة طبقات مع عمليات التنشيط المطبقة) مسؤولاً عن تعلم التعيين، M، من المدخلات x إلى المخرجات y.

$$M(x) = y$$

بدلاً من تعلم التعيين المباشر من x إلى y، يمكننا بدلاً من ذلك تحديد دالة متبقية.

$$F(x) = M(x) - x$$

ينظر هذا إلى الفرق بين التعيين المطبق على x والإدخال الأصلي x. عادةً ما تكون F(x) عبارة عن طبقتين تلافيفيتين + طبقة تسوية وReLU بينهما. يجب أن تحتوي هذه الطبقات التلافيفية على نفس عدد المدخلات مثل المخرجات. ويمكن بعد ذلك كتابة هذا التعيين على النحو التالي؛ دالة للدالة المتبقية والمدخلات x.

$$M(x) = F(x) + x$$

يمكنك قراءة المزيد عن التعلم المتبقي العميق هنا. فيما يلي مقتطف التعليمات البرمجية لتطبيق الكتل المتبقية.

```
class ResidualBlock(nn.Module):
"""Defines a residual block.
This adds an input x to a convolutional layer (applied to x)
with the same size input and output.
These blocks allow a model to learn an effective
transformation from one domain to another.
"""
def __init__(self, conv_dim):
super(ResidualBlock, self).__init__()
# conv_dim = number of inputs

# define two convolutional layers + batch normalization that
```

```
will act as our residual function, F(x)
# layers should have the same shape input as output; I
suggest a kernel_size of 3
self.layer_1 = conv(conv_dim,conv_dim,3,1,1,batch_norm =
True)
self.layer_2 = conv(conv_dim,conv_dim,3,1,1,batch_norm =
True)

def forward(self, x):
# apply a ReLu activation the outputs of the first layer
# return a summed output, x + resnet_block(x)
out_1 = F.relu(self.layer_1(x))
out_2 = x + self.layer_2(out_1)

return out_2
```

المولد

يتكون Generator G_xtoy وG_ytox من المشفر encoder، وconv net تحول الصورة إلى تمثيل صغير للميزات، ومفكك شفرة decoder، وشبكة transpose_conv مسؤولة عن تحويل تمثيل الميزة إلى صورة محولة transformed image. فيما يلي مقتطف التعليمات البرمجية لتنفيذ المولد.

```
def deconv(in_channels, out_channels, kernel_size, stride=2,
padding=1, batch_norm=True):
"""Creates a transpose convolutional layer, with optional
batch normalization.
"""
layers = []
# append transpose conv layer
layers.append(nn.ConvTranspose2d(in_channels, out_channels,
kernel_size, stride, padding, bias=False))
# optional batch norm layer
if batch_norm:
layers.append(nn.BatchNorm2d(out_channels))
return nn.Sequential(*layers)

class CycleGenerator(nn.Module):

def __init__(self, conv_dim=64, n_res_blocks=6):
super(CycleGenerator, self).__init__()

# 1. Define the encoder part of the generator
self.layer_1 = conv(3,conv_dim,4)
self.layer_2 = conv(conv_dim,conv_dim*2,4)
self.layer_3 = conv(conv_dim*2,conv_dim*4,4)
# 2. Define the resnet part of the generator
layers = []
for n in range(n_res_blocks):
```

```
layers.append(ResidualBlock(conv_dim*4))
self.res_blocks = nn.Sequential(*layers)
# 3. Define the decoder part of the generator
self.layer_4 = deconv(conv_dim*4,conv_dim*2,4)
self.layer_5 = deconv(conv_dim*2,conv_dim,4)
self.layer_6 = deconv(conv_dim,3,4,batch_norm = False)

def forward(self, x):
"""Given an image x, returns a transformed image."""
# define feedforward behavior, applying activations as
necessary

out = F.relu(self.layer_1(x))
out = F.relu(self.layer_2(out))
out = F.relu(self.layer_3(out))

out = self.res_blocks(out)

out = F.relu(self.layer_4(out))
out = F.relu(self.layer_5(out))
out = F.tanh(self.layer_6(out))

return out
```

الشرح:

- تتكون المعمارية التالية من ثلاث طبقات تلافيفية للمشفر وثلاث طبقات تلافيفية منقولة لمفكك الشفرة، وكلاهما متصلان باستخدام سلسلة من الكتل المتبقية (في حالتنا 6).
- جميع الطبقات التلافيفية يتبعها تسوية بالدفعات.
- جميع الطبقات التلافيفية المنقولة، باستثناء الطبقة الأخيرة، يتبعها تسوية بالدفعات.
- بالنسبة للوحدات المخفية، يتم استخدام دالة التنشيط ReLU، باستثناء الطبقة الأخيرة حيث نستخدم دالة التنشيط tanh.
- يعتمد عدد خرائط الميزات بعد كل التفاف في المشفر ومفكك الشفرة على المعلمة conv_dim.

يتمتع كل من G_xtoy وG_ytox بنفس المعمارية، لذلك نحتاج فقط إلى تحديد فئة واحدة، ثم إنشاء مولدين لاحقًا.

عملية التدريب

تشتمل عملية التدريب على تحديد دوال الخطأ واختيار المحسن وأخيراً تدريب النموذج.

خطأ المميز والمولد

لقد رأينا أن شبكات GAN العادية تتعامل مع المُميِّز كمصنف مع دالة خطأ الإنتروبيا السيني sigmoid cross-entropy loss. ومع ذلك، قد تؤدي دالة الخطأ هذه إلى مشكلة التدرج المتلاشي vanishing gradient أثناء عملية التعلم. للتغلب على هذه المشكلة، سنستخدم دالة خطأ المربعات الصغرى least-squares loss function للمميز. غالبًا ما يُشار إلى هذه المعمارية باسم شبكات GAN ذات المربعات الصغرى Least Square GANs، ويمكنك قراءة المزيد عنها من الورقة الأصلية لشبكات LSGAN.

خطأ المميز

ستكون اخطاء المميز هي متوسط الأخطاء المربعة mean squared errors بين مخرجات المميز،في ضوء الصورة، والقيمة المستهدفة، 0 أو 1، اعتمادًا على ما إذاكان ينبغي تصنيف تلك الصورة على أنها مزيفة أو حقيقية. على سبيل المثال، بالنسبة لصورة حقيقية، x، يمكننا تدريب D_x من خلال النظر في مدى قربها من التعرف على الصورة x باعتبارها حقيقية باستخدام متوسط الخطأ المربع:

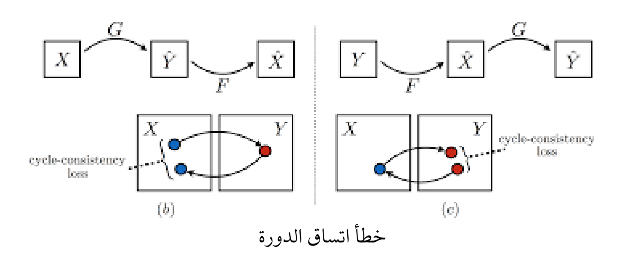
out = D_x(x)

real_error = torch.mean((out-1)²) (for Pytorch)

خطأ المولد

في هذا، سنقوم بإنشاء صور مزيفة تبدو وكأنها تنتمي إلى المجال X ولكنها تعتمد على صور من المجال Y، والعكس صحيح. سنقوم بحساب الخطأ الحقيقي لتلك الصور التي تم إنشاؤها من خلال النظر إلى مخرجات المميز عند تطبيقها على هذه الصور المزيفة.

بالإضافة إلى الخطأ العدائي adversarial loss، سيشمل حد خطأ المولد Generator loss خطأ اتساق الدورة cycle consistency loss. تعد هذه الخطأ مقياسًا لمدى جودة الصورة المعاد بناؤها عند مقارنتها بالصورة الأصلية. على سبيل المثال، لدينا صورة مزيفة x^ وصورة حقيقية y، يمكننا إنشاء y^ من x^ بمساعدة G_xtoy(G_xtoy(x^) = y^). هنا سيكون خطأ اتساق الدورة هو الفرق المطلق بين الصورة الأصلية والصورة المعاد بناؤها.



خطأ اتساق الدورة

فيما يلي مقتطف الكود لتحديد الاخطاء.

```
def real_mse_loss(D_out):
# how close is the produced output from being "real"?
return torch.mean((D_out - 1)**2)


def fake_mse_loss(D_out):
# how close is the produced output from being "fake"?
return torch.mean(D_out**2)

def cycle_consistency_loss(real_im, reconstructed_im,
lambda_weight):
# calculate reconstruction loss
# return weighted loss
loss = torch.mean(torch.abs(real_im - reconstructed_im))
return loss*lambda_weight
```

في خطأ اتساق الدورة، يكون حد لامدا lambda هو معلمة الوزن التي ستقوم بوزن متوسط الخطأ المطلق في الدفعة. من المستحسن إلقاء نظرة على ورقة CycleGAN الأصلية للحصول على قيمة أولية لوزن lambda.

المحسن

بالنسبة لـ CycleGAN، قمنا بتحديد ثلاثة أدوات تحسين للمولدات (G_xtoy وG_ytox) وD_x وD_y. بالنسبة لجميع أدوات تحسين الأداء، فإننا نستخدم Adam. يتم اختيار جميع المعلمات الفائقة للقيم من ورقة CycleGAN الأصلية.

```
# hyperparams for Adam optimizers
lr= 0.0002
beta1= 0.5
beta2= 0.999

g_params = list(G_XtoY.parameters()) +
list(G_YtoX.parameters()) # Get generator parameters

# Create optimizers for the generators and discriminators
g_optimizer = optim.Adam(g_params, lr, [beta1, beta2])
d_x_optimizer = optim.Adam(D_X.parameters(), lr, [beta1,
beta2])
d_y_optimizer = optim.Adam(D_Y.parameters(), lr, [beta1,
beta2])
```

التدريب

عندما يتدرب CycleGAN، ويرى مجموعة واحدة من الصور الحقيقية من المجموعتين X وY، فإنه يتدرب عن طريق تنفيذ الخطوات التالية:

للمميز:

- حساب خطأ Discriminator D_x على الصور الحقيقية.
- قم بإنشاء صور مزيفة بمساعدة G_ytox باستخدام صور من المجموعة Y، ثم احسب الخطأ المزيف لـ D_x.
- حساب الخطأ الإجمالي وإجراء الانتشار الخلفي والتحسين. افعل الشيء نفسه مع D_y وقم بتبديل المجال الخاص بك.

للمولد:

- قم بإنشاء صور مزيفة تشبه المجال X بناءً على صور حقيقيةفي المجال Y، ثم قم بحساب خطأ المولد بناءً على كيفية استجابة D_x لـ X المزيف.
- قم بإنشاء صور Y^ معاد بناؤها بناءً على صور X المزيفةفي الخطوة 1.
- حساب خطأ اتساق الدورة على صور Y المعاد بناؤها والحقيقية.
- كرر الخطوات من 1 إلى 4، مع تبديل المجالات فقط وإضافة كافة اخطاء المولد وإجراء الانتشار الخلفي والتحسين.

إليك مقتطف الكود للقيام بذلك.

```
def training_loop(dataloader_X, dataloader_Y,
test_dataloader_X, test_dataloader_Y,
n_epochs=1000):

print_every=10

# keep track of losses over time
losses = []

test_iter_X = iter(test_dataloader_X)
test_iter_Y = iter(test_dataloader_Y)

# Get some fixed data from domains X and Y for sampling.
These are images that are held
# constant throughout training, that allow us to inspect the
model's performance.
fixed_X = test_iter_X.next()[0]
fixed_Y = test_iter_Y.next()[0]
fixed_X = scale(fixed_X) # make sure to scale to a range -1
to 1
fixed_Y = scale(fixed_Y)

# batches per epoch
iter_X = iter(dataloader_X)
```

```
iter_Y = iter(dataloader_Y)
batches_per_epoch = min(len(iter_X), len(iter_Y))

for epoch in range(1, n_epochs+1):

# Reset iterators for each epoch
if epoch % batches_per_epoch == 0:
iter_X = iter(dataloader_X)
iter_Y = iter(dataloader_Y)

images_X, _ = iter_X.next()
images_X = scale(images_X) # make sure to scale to a range -
1 to 1

images_Y, _ = iter_Y.next()
images_Y = scale(images_Y)

# move images to GPU if available (otherwise stay on CPU)
device = torch.device("cuda:0" if torch.cuda.is_available()
else "cpu")
images_X = images_X.to(device)
images_Y = images_Y.to(device)


# ============================================
# TRAIN THE DISCRIMINATORS
# ============================================

## First: D_X, real and fake loss components ##

# 1. Compute the discriminator losses on real images
d_x_optimizer.zero_grad()
real_D_loss = real_mse_loss(D_X(images_X))
# 3. Compute the fake loss for D_X
fake_D_loss = fake_mse_loss(D_X(G_YtoX(images_Y)))
# 4. Compute the total loss and perform backprop
d_x_loss = real_D_loss + fake_D_loss
d_x_loss.backward()
d_x_optimizer.step()

## Second: D_Y, real and fake loss components ##
d_y_optimizer.zero_grad()
real_D_y_loss = real_mse_loss(D_Y(images_Y))

fake_D_y_loss = fake_mse_loss(D_Y(G_XtoY(images_X)))

d_y_loss = real_D_y_loss + fake_D_y_loss
d_y_loss.backward()
d_y_optimizer.step()
```

```
# ============================================
# TRAIN THE GENERATORS
# ============================================

## First: generate fake X images and reconstructed Y images
##
g_optimizer.zero_grad()
# 1. Generate fake images that look like domain X based on
real images in domain Y
out_1 = G_YtoX(images_Y)
# 2. Compute the generator loss based on domain X
loss_1 = real_mse_loss(D_X(out_1))
# 3. Create a reconstructed y
out_2 = G_XtoY(out_1)
# 4. Compute the cycle consistency loss (the reconstruction
loss)
loss_2 = cycle_consistency_loss(real_im = images_Y,
reconstructed_im = out_2, lambda_weight=10)

## Second: generate fake Y images and reconstructed X images
##
out_3 = G_XtoY(images_X)
# 5. Add up all generator and reconstructed losses and
perform backprop
loss_3 = real_mse_loss(D_Y(out_3))
out_4 = G_YtoX(out_3)
loss_4 = cycle_consistency_loss(real_im = images_X,
reconstructed_im = out_4, lambda_weight=10)

g_total_loss = loss_1 + loss_2 + loss_3 + loss_4
g_total_loss.backward()
g_optimizer.step()

# Print the log info
if epoch % print_every == 0:
# append real and fake discriminator losses and the
generator loss
losses.append((d_x_loss.item(), d_y_loss.item(),
g_total_loss.item()))
print('Epoch [{:5d}/{:5d}] | d_X_loss: {:6.4f} | d_Y_loss:
{:6.4f} | g_total_loss: {:6.4f}'.format(
epoch, n_epochs, d_x_loss.item(), d_y_loss.item(),
g_total_loss.item()))


sample_every=100
# Save the generated samples
if epoch % sample_every == 0:
G_YtoX.eval() # set generators to eval mode for sample
```

```
generation
G_XtoY.eval()
save_samples(epoch, fixed_Y, fixed_X, G_YtoX, G_XtoY,
batch_size=16)
G_YtoX.train()
G_XtoY.train()

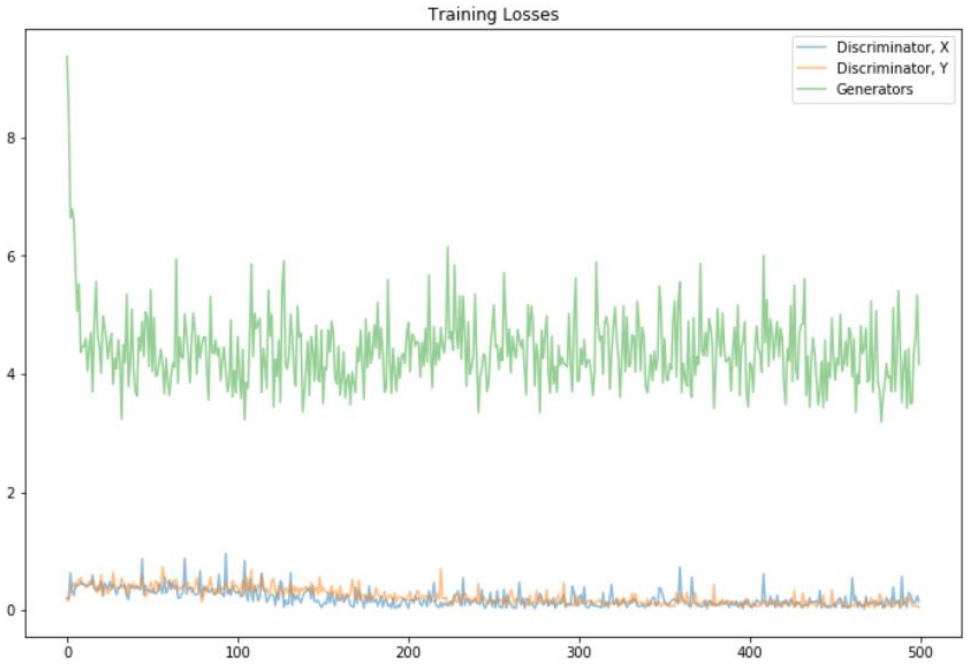
# uncomment these lines, if you want to save your model
# checkpoint_every=1000
# # Save the model parameters
# if epoch % checkpoint_every == 0:
# checkpoint(epoch, G_XtoY, G_YtoX, D_X, D_Y)

return losses
```

يتم تنفيذ التدريب على مدى 5000 فترة epochs باستخدام وحدة معالجة الرسوميات GPU، ولهذا السبب اضطررت إلى نقل النموذج والمدخلات الخاصة بي من وحدة المعالجة المركزية CPU إلى وحدة معالجة الرسوميات GPU.

النتائج

فيما يلي مخطط اخطاء التدريب للمولد والمميز المسجل بعد كل فترة.



يمكننا أن نلاحظ أن المولدات تبدأ بخطأ كبير جدًا، ولكن مع مرور الوقت تبدأفي إنتاج ترجمات جيدة للصور، مما يساعدفي تقليل الخطأ.

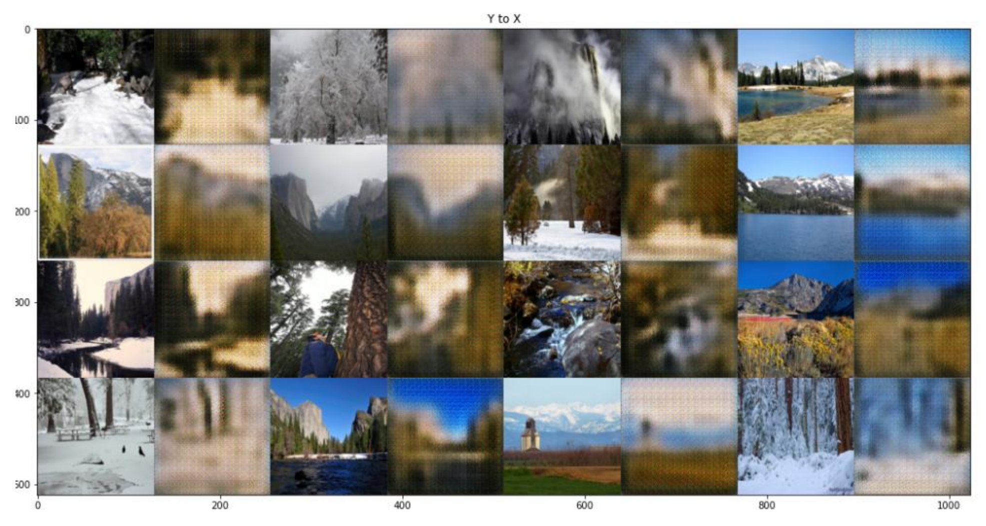
يُظهر كلا خطأين المميز تقلبًا طفيفًا جدًا في الخطأ. ولكن بحلول نهاية 5000 فترة epoch، يمكننا أن نرى أن كلا من أخطاء المميز قد انخفضت، مما يجبر المولدات على القيام بترجمات صور أكثر واقعية.

- تصور العينات.

بعد 100 تكرار:

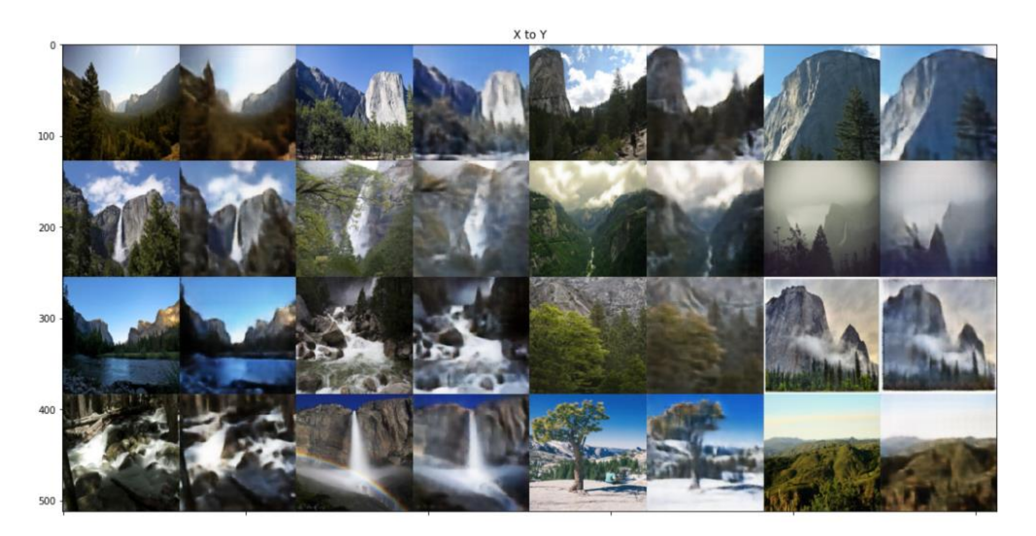


الترجمة من X إلى Y بعد 100 تكرار



الترجمة من Y إلى X بعد 100 تكرار

بعد 5000 تكرار:



الترجمة من X إلى Y بعد 5000 تكرار



الترجمة من Y إلى X بعد 5000 تكرار

يمكننا أن نلاحظ أن نماذج CycleGAN تنتج صورًا منخفضة الدقة، وهذا مجال بحث مستمر، ويمكنك قراءة المزيد عن الصيغة عالية الدقة التي تستخدم مولدات متعددة بالنقر هنا.

يواجه هذا النموذج صعوبةفي مطابقة الألوان تمامًا. وذلك لأنه إذا كان G_xtoy وG_ytox قد يغيران لون الصورة؛ قد لا يتأثر خطأ اتساق الدورة ويمكن أن يظل صغيرًا. يمكنك اختيار تقديم مصطلح خطأ جديد يعتمد على اللون يقارن بين G_ytox(y) وy وG_xtoy(x) وx، ولكن يصبح هذا بعد ذلك أسلوبًا للتعلم تحت الإشراف. ومع ذلك، تمكنت CycleGAN من القيام بترجمات مرضية.

المصدر:

https://towardsdatascience.com/image-to-image-translation-using-cyclegan-model-d58cfff04755

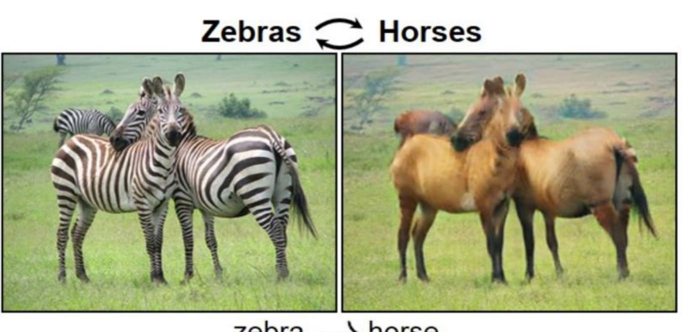
14) تلوين الصور بالأبيض والأسود باستخدام شبكة الخصومة التوليدية GAN في TensorFlow Colorizing B/W Images with GANs in TensorFlow

تُعد شبكات GAN واحدة من أكثر المواضيع إثارة للاهتمامفي التعلم الآلي اليوم. لقد تم استخدامهافي عدد من المشكلات (وليس فقط لإنشاء أرقام MNIST!) وكان أداؤها جيداً للغايةفي كل حالة. تتكون شبكة GAN (شبكة الخصومة العامة General Adversarial Network) من مولد generator ومميز discriminator، يتنافسان ضد بعضهما البعض لتحقيق نتائج مذهلة. سنتبع هنا منهجًا رياضيًا لفهم شبكة GAN ودوال الخطأ loss functions الخاصة بها. وبما أن فكرة تدريب GAN تأتي من نظرية الالعاب game theory، فسنلقي نظرة سريعة على استراتيجية التحسين Minimax أيضًا.

في هذه المقالة، سوف نستكشف شبكات GAN لتلوين الصور بالأبيض والأسود ونتعرف أيضًا على دوال الخطأ المطلوبة لنموذجنا. لذا، استعد لبعض شبكات GAN!

"[شبكات GAN] هي الفكرة الأكثر إثارة للاهتمام في السنوات العشر الماضية في مجال التعلم الآلي" – يان ليكون، مدير Facebook AI

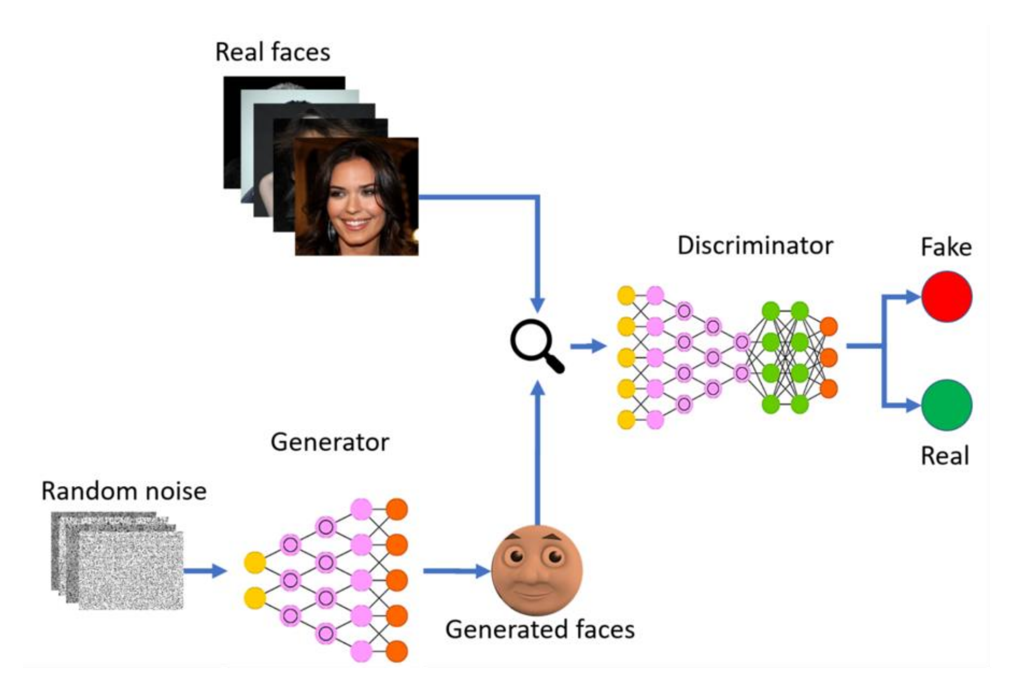
تتمتع شبكات GAN ببعض التطبيقات المدهشة، مثل تحويل الحصان إلى حمار وحشي، كما هو موضح أدناه.






horse → zebra

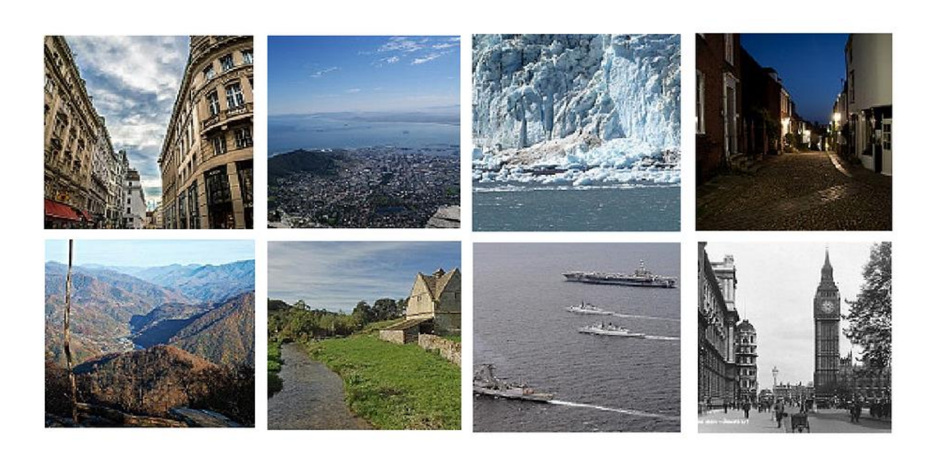
فيما يلي المعمارية الأساسية لشبكات GAN المستخدمةفي إنشاء وجوه بشرية واقعية:



لن تتمكن هذه المقالة إلا من إعطائك لمحة عن كيفية عمل شبكات GAN، حيث سنركز أكثر على حالة الاستخدام بدلاً من الشرح الكامل لكيفية عمل شبكات GAN.

لقد حاولت تلوين الصور باستخدام AutoEncoders من قبل، لكن النتائج لم تكن بالمستوى المطلوب. ظهر لون واحدفي الصورة بأكملها بظلال أو صبغات مختلفة. كود المشروع متاح هنا ->

البيانات والكود



ستتألف مجموعة البيانات الخاصة بنا من 3000 صورة RGB من مجالات مختلفة (الجبال والغابات والمدن وما إلى ذلك). يمكنك تحميله من هنا.في Colab notebook، سنقوم بتحويل صور RGB هذه إلى تدرج رمادي باستخدام PIL والتي ستكون بمثابة تسميات labels لنموذجنا.

يمكن العثور على تطبيق TensorFlow لهذا المشروع في دفتر Colab notebook هذا.

المولد

أول شيء ستحتاجه GAN لدينا هو المولد generator. سوف يأخذ هذا المولد صورة ذات تدرج رمادي أو أبيض وأسود، ويخرج صورة RGB. سيكون لدى المولد الخاص بنا معمارية مشفر ومفكك شفرة encoder-decoder structure مع طبقات موضوعة بشكل متماثل، تمامًا مثل UNet.

سيلتقط المشفر encoder صورة ذات تدرج رمادي وينتج تمثيلًا كامنًا لها (يُسمى أيضًا تمثيل عنق الزجاجة bottleneck representation). تتمثل مهمة مفكك الشفرة decoder في إنتاج صورة RGB من خلال توسيع هذا التمثيل الكامن latent representation. يتم استخدام هذا الأسلوب من قبل معظم شبكات الترميز التلقائي autoencoders بالإضافة إلى هياكل المشفر ومفكك الشفرة الأخرى.

أثناء إنشاء صورة RGB من التمثيل الكامن، قد تكون بعض التفاصيل الدقيقة مفقودة. سيكون من المثير للاهتمام ملاحظة النتائج إذا أمكن أن تأتي المعلومات مباشرة من المشفر إلى مفكك الشفرة. هنا يأتي دور تخطي الاتصالات الاتصالات skip-connections في الصورة.

تقوم اتصالات التخطي بإحضار مخرجات طبقة الالتفاف convolution layer (الموجودة في المشفر) إلى مفكك الشفرة، حيث يتم ربطها مع المخرجات السابقة لمفكك الشفرة نفسه.

```
def get_generator_model():

    inputs = tf.keras.layers.Input( shape=( img_size ,
img_size , 1 ) )

    conv1 = tf.keras.layers.Conv2D( 16 , kernel_size=( 5 , 5
) , strides=1 , dilation_rate=4 )( inputs )
    conv1 = tf.keras.layers.LeakyReLU()( conv1 )
    conv1 = tf.keras.layers.Conv2D( 32 , kernel_size=( 3 , 3
) , strides=1 , dilation_rate=2 )( conv1 )
    conv1 = tf.keras.layers.LeakyReLU()( conv1 )
    conv1 = tf.keras.layers.Conv2D( 32 , kernel_size=( 3 , 3
) , strides=1)( conv1 )
    conv1 = tf.keras.layers.LeakyReLU()( conv1 )

    conv2 = tf.keras.layers.Conv2D( 32 , kernel_size=( 5 , 5
) , strides=1)( conv1 )
```

```
    conv2 = tf.keras.layers.LeakyReLU()( conv2 )
    conv2 = tf.keras.layers.Conv2D( 64 , kernel_size=( 3 , 3
) , strides=1 )( conv2 )
    conv2 = tf.keras.layers.LeakyReLU()( conv2 )
    conv2 = tf.keras.layers.Conv2D( 64 , kernel_size=( 3 , 3
) , strides=1 )( conv2 )
    conv2 = tf.keras.layers.LeakyReLU()( conv2 )

    conv3 = tf.keras.layers.Conv2D( 64 , kernel_size=( 5 , 5
) , strides=1 )( conv2 )
    conv3 = tf.keras.layers.LeakyReLU()( conv3 )
    conv3 = tf.keras.layers.Conv2D( 128 , kernel_size=( 3 ,
3 ) , strides=1 )( conv3 )
    conv3 = tf.keras.layers.LeakyReLU()( conv3 )
    conv3 = tf.keras.layers.Conv2D( 128 , kernel_size=( 3 ,
3 ) , strides=1 )( conv3 )
    conv3 = tf.keras.layers.LeakyReLU()( conv3 )

    bottleneck = tf.keras.layers.Conv2D( 128 , kernel_size=(
3 , 3 ) , strides=1 , activation='relu' , padding='same' )(
conv3 )

    concat_1 = tf.keras.layers.Concatenate()( [ bottleneck ,
conv3 ] )
    conv_up_3 = tf.keras.layers.Conv2DTranspose( 128 ,
kernel_size=( 3 , 3 ) , strides=1 , activation='relu' )(
concat_1 )
    conv_up_3 = tf.keras.layers.Conv2DTranspose( 128 ,
kernel_size=( 3 , 3 ) , strides=1 , activation='relu' )(
conv_up_3 )
    conv_up_3 = tf.keras.layers.Conv2DTranspose( 64 ,
kernel_size=( 5 , 5 ) , strides=1 , activation='relu' )(
conv_up_3 )

    concat_2 = tf.keras.layers.Concatenate()( [ conv_up_3 ,
conv2 ] )
    conv_up_2 = tf.keras.layers.Conv2DTranspose( 64 ,
kernel_size=( 3 , 3 ) , strides=1 , activation='relu' )(
concat_2 )
    conv_up_2 = tf.keras.layers.Conv2DTranspose( 64 ,
kernel_size=( 3 , 3 ) , strides=1 , activation='relu' )(
conv_up_2 )
    conv_up_2 = tf.keras.layers.Conv2DTranspose( 32 ,
kernel_size=( 5 , 5 ) , strides=1 , activation='relu' )(
conv_up_2 )

    concat_3 = tf.keras.layers.Concatenate()( [ conv_up_2 ,
conv1 ] )
```

```
    conv_up_1 = tf.keras.layers.Conv2DTranspose( 32 ,
kernel_size=( 3 , 3 ) , strides=1 , activation='relu')(
concat_3 )
    conv_up_1 = tf.keras.layers.Conv2DTranspose( 32 ,
kernel_size=( 3 , 3 ) , strides=1 , activation='relu' ,
dilation_rate=2 )( conv_up_1 )
    conv_up_1 = tf.keras.layers.Conv2DTranspose( 3 ,
kernel_size=( 5 , 5 ) , strides=1 , activation='relu' ,
dilation_rate=4 )( conv_up_1 )

    model = tf.keras.models.Model( inputs , conv_up_1 )
    return model
```

- يمكنك ملاحظة تخطي الاتصالات في الأسطر 28 و33 و38 في المقتطف أعلاه.

المميز

سيكون المُميز discriminator الخاص بنا عبارة عن شبكة CNN القياسية التي نستخدمها للتصنيف. سوف يأخذ صورة ويخرج احتمالية ما إذا كانت الصورة المعطاة أصلية أو إذا تم إنشاؤها (بواسطة المولد).

```
def get_discriminator_model():
    layers = [
        tf.keras.layers.Conv2D( 32 , kernel_size=( 7 , 7 ) ,
strides=1 , activation='relu' , input_shape=( 120 , 120 , 3
) ),
        tf.keras.layers.Conv2D( 32 , kernel_size=( 7, 7 ) ,
strides=1, activation='relu'  ),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D( 64 , kernel_size=( 5 , 5 ) ,
strides=1, activation='relu'  ),
        tf.keras.layers.Conv2D( 64 , kernel_size=( 5 , 5 ) ,
strides=1, activation='relu'  ),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D( 128 , kernel_size=( 3 , 3 )
, strides=1, activation='relu'  ),
        tf.keras.layers.Conv2D( 128 , kernel_size=( 3 , 3 )
, strides=1, activation='relu'  ),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D( 256 , kernel_size=( 3 , 3 )
, strides=1, activation='relu'  ),
        tf.keras.layers.Conv2D( 256 , kernel_size=( 3 , 3 )
, strides=1, activation='relu'  ),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense( 512, activation='relu'  )  ,
        tf.keras.layers.Dense( 128 , activation='relu' ) ,
        tf.keras.layers.Dense( 16 , activation='relu' ) ,
        tf.keras.layers.Dense( 1 , activation='sigmoid' )
```

```
    ]
    model = tf.keras.models.Sequential( layers )
    return model
```

الرياضيات

أعلم أن الرياضيات تصبح مخيفة، خاصةفي التعلم الآلي، لكن لا داعي للقلق، سأبقي الأمور بسيطة قدر الإمكان. لنفترض أن لدينا عينة (x, y) من مجموعة البيانات الخاصة بنا. هنا تمثل x صورة ذات تدرج رمادي وستكون y هي نفس الصورة ولكن مع الألوان، أي بتنسيق RGB. أشكال x و y موضحة أدناه.

$$x \rightarrow (B, 120, 120, 1) \quad y \rightarrow (B, 120, 120, 3)$$

هنا يمثل "B" حجم الدفعة batch size.

نحن نمثل المولد بالرمز G والمميز بالرمز D. وفي خطوة واحدة، سنقوم بتشغيل المولد مرة والمميز مرتين.

$$y_p = G(\ x\ )$$

$$P(\ real \mid y\ ) = D(\ y\ )$$
$$P(\ real \mid y_p\ ) = D(\ y_p\ )$$

هنا يمثل y_p الصورة التي تم إنشاؤها. P(real | y) هو احتمال أن تكون الصورة المعطاة y هي الصورة الموجودةفي البيانات. هنا، تشير كلمة "real" إلى أن الصورة لم يتم إنشاؤها.

يمكننا أن نعتبر y حقيقية و y_p كصورة تم إنشاؤها/مزيفة من المولد. قد نستخدم هذه المصطلحات كما ستجدفي معظم الموارد التي تشرح شبكات GAN.

الآن دعونا نلقي نظرة على دوال الخطأ. أولاً، بالنسبة للمولد، سنستخدم دالة الخطأ MSE. يتم استخدامه بشكل عامفي الانحدار، ولكن يمكننا استخدامه أيضًافي حالتنا.

$$L_G = MSE(\ x\ ,\ y_p\ )$$

في المعادلة أعلاه، y_p هي الصورة التي تم إنشاؤها وx هي الصورة المدخلة. دوال الخطأ للمميز مبينة أدناه. نحن نستخدم خطأ الإنتروبيا المتقاطعة الثنائية binary cross-entropy loss لكلا مخرجات المُميِّز.

$$L_D^y = \log(\ D(\ y\ )\ )$$

$$L_D^{y_p} = \log(\ 1 - \ D(\ y_p\ )\ )$$

نضيف دوال الخطأ هذه للحصول على التعبير النهائي لدالة الخطأ للمميز:

$$L_D = L_D^y + L_D^{y_p}$$

$$= \log(\ D(\ y\ )\ ) + \log(\ 1 - \ D(\ y_p\ )\ )$$

- يشير Minimax إلى استراتيجية التحسين في الألعاب القائمة على تبادل الأدوار بين لاعبين لتقليل الخطأ أو التكلفة لأسوأ حالة للاعب الآخر. هنا، المولد والمميز هما اللاعبان اللذان يتنافسان ضد بعضهما البعض.
- بالنسبة للمميز، فإن تعظيم خطأهما يعني تصنيف الصور التي تم إنشاؤها (y_p) بدقة بالإضافة إلى إنتاج احتمالية جيدة (أقرب إلى 1.0) للصور (y) من مجموعة البيانات.

$$\max_D \left[\ \log(\ D(\ y\ )\ ) + \log(\ 1 - \ D(\ y_p\ )\ )\ \right]$$

- المولد، من خلال تقليل خطأه، يعمل على تحسين نفسه إلى الحد الذي يمكنه من خداع المُميِّز. خداع المُميِّز يعني أن المُميِّز سينتج احتمالات (أقرب إلى 1.0) حتى بالنسبة للصور التي تم إنشاؤها (y_p).

$$\min_G\ MSE(\ x\ ,\ G(\ x\ )\ )$$

سنقوم بتدريب المميز بطريقة تجعل احتمالات الإخراج أقرب إلى 1.0 للصور الحقيقية (من مجموعة البيانات الخاصة بنا) واحتمالات الإخراج أقرب إلى 0.0 للصور القادمة من المولد.

إذا كان المميز "ذكي" بدرجة كافية، فسوف تنتج احتمالات أقرب إلى 1.0 للصور الحقيقية (القادمة من مجموعة البيانات الخاصة بنا). لذلك، نحن نقوم بتدريب المولد الخاص بنا على صياغة مثل هذه الصور الواقعية التي ستجعل احتمالات إخراج المُميِّز أقرب إلى 1.0 حتى عندما تكون الصور مزيفة (ليس من مجموعة البيانات الخاصة بنا، ولكن من المولد).

ستكون دالة الخطأ النهائي لدينا هي:

$$\min_G \max_D \left[\ \log(\ D(\ y\ )\ ) + \log(\ 1 - \ D(\ y_p\ )\ )\ +\ MSE(\ x\ ,\ G(x)\ )\ \right]$$

هذا كل شيء! سنتوجه الآن نحو التعليمات البرمجية لتدريب GAN.

الكود

لقد رأينا تنفيذ Keras للمولد والمميز في المقتطفين 1 و2. الآن دعونا نلقي نظرة على تنفيذ دوال الخطأ.

```
cross_entropy = tf.keras.losses.BinaryCrossentropy()
mse = tf.keras.losses.MeanSquaredError()

def discriminator_loss(real_output, fake_output):
```

```
    real_loss = cross_entropy(tf.ones_like(real_output) -
tf.random.uniform( shape=real_output.shape , maxval=0.1 ) ,
real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output) +
tf.random.uniform( shape=fake_output.shape , maxval=0.1  ) ,
fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output , real_y):
    real_y = tf.cast( real_y , 'float32' )
    return mse( fake_output , real_y )

generator_optimizer = tf.keras.optimizers.Adam( 0.001 )
discriminator_optimizer = tf.keras.optimizers.Adam( 0.001 )
```

هل لاحظت شيئًا مختلفًافي المقتطف أعلاه،في السطرين 5 و6؟

نحن نقوم بإضافة/طرح قيم عشوائية صغيرة من tf.ones وtf.zeros. لذا، بدلاً من استخدام التصنيفات الثابتة مثل 1 و0، نستخدم تسميات مشوشة مثل 0.12 أو 0.99. يساعد هذا المُميِّز على التعلم بشكل أفضل وإلا فإنه سيقترب من 1 أو 0في الفترات الأولية ولن يحدث أي تعلم.

نستخدم مُحسِّن Adam لكل من المولد والمميز بمعدل تعلم learning rate قدره 0.001.

التالي يأتي حلقة التدريب. ستقوم حلقة التدريب بإنشاء تنبؤات، سواء من المولد أو المُميز، وحساب الاخطاء، وتحسين كلا النموذجين.

```
@tf.function
def train_step( input_x , real_y ):

    with tf.GradientTape() as gen_tape, tf.GradientTape() as
disc_tape:
        # Generate an image -> G( x )
        generated_images = generator( input_x ,
training=True)
        # Probability that the given image is real -> D( x )
        real_output = discriminator( real_y, training=True)
        # Probability that the given image is the one
generated -> D( G( x ) )
        generated_output = discriminator(generated_images,
training=True)

        # MSE
        gen_loss = generator_loss( generated_images , real_y
)
        # Log loss for the discriminator
        disc_loss = discriminator_loss( real_output,
generated_output )
```

```
    #tf.keras.backend.print_tensor( tf.keras.backend.mean(
gen_loss ) )
    #tf.keras.backend.print_tensor( gen_loss + disc_loss )

    # Compute the gradients
    gradients_of_generator = gen_tape.gradient(gen_loss,
generator.trainable_variables)
    gradients_of_discriminator =
disc_tape.gradient(disc_loss,
discriminator.trainable_variables)

    # Optimize with Adam

generator_optimizer.apply_gradients(zip(gradients_of_generat
or, generator.trainable_variables))

discriminator_optimizer.apply_gradients(zip(gradients_of_dis
criminator, discriminator.trainable_variables))
```

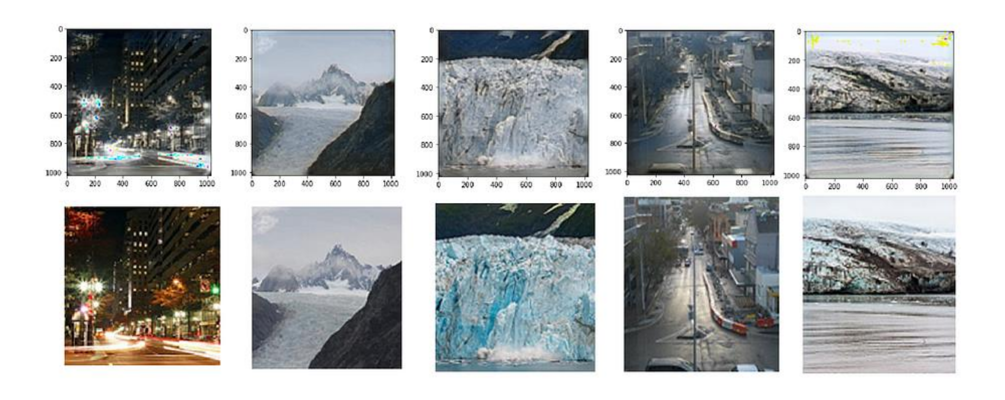
نحن الآن على استعداد لبدء التدريب. تتم طباعة قيم الخطأ لكل تمريرة للأمام.

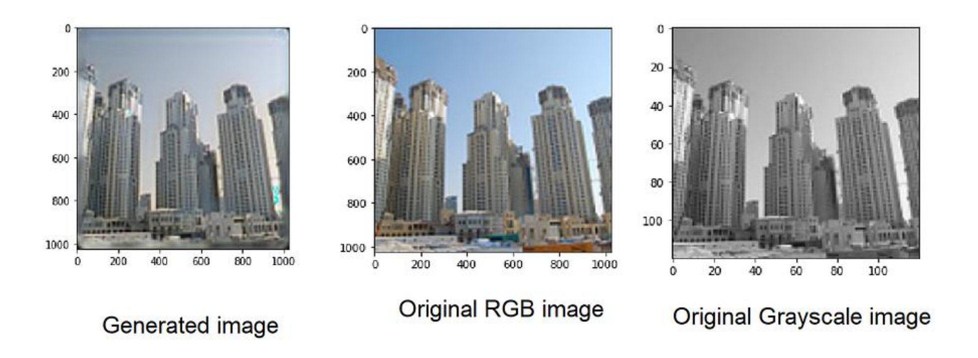
```
num_epochs = 60

for e in range( num_epochs ):
    print( 'Epoch ' , e )
    for ( x , y ) in dataset:
        # Here ( x , y ) represents a batch from our
training dataset.
        train_step( x , y )
```

النتائج

النتائج جيدة جداً وتظهر القوة المذهلة لشبكات GAN. لكنك ستشاهد بعض الإزعاج (البقع ذات اللون الأسود/الأصفر المتميزة عن خلفيتها)في الصور أدناه.





Generated image

Original RGB image

Original Grayscale image

لقد حصلنا أيضًا على بعض النتائج المفاجئة حيث تم تحويل النهار إلى المساء ـ ولم ندرب النموذج على ذلك!

Generated

Original

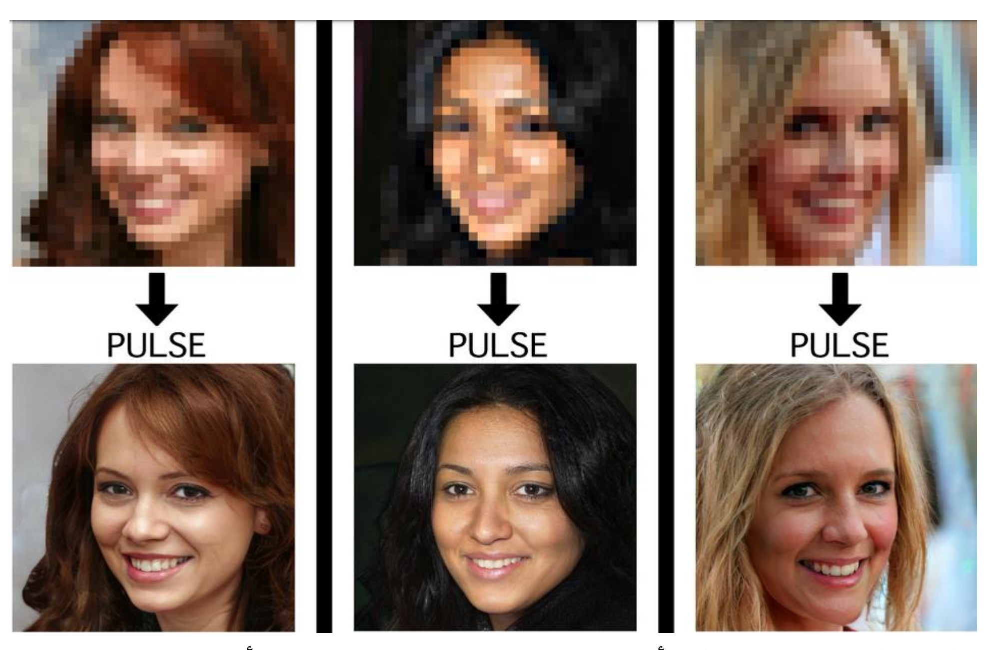
Generated

Original

الاستنتاج

حقًا، تتمتع شبكات GAN بالقدرة على تغيير وجه التعلم الآلي. وكما قلت سابقًا، فهي مرنة ويمكن استخدامها لحل المشكلات المختلفة. هل تتساءل أين يمكننا استخدام شبكات GAN؟

تم استخدام شبكات GAN للحصول على دقة فائقة للصور. وهنا نقوم بتحويل صورة منخفضة الدقة إلى صورة عالية الدقة كما هو موضح أدناه،



يمكن لشبكات GAN (بشكل أكثر دقة CycleGANs) إنشاء لوحات وأعمال فنية تشبه الإنسان، كما هو موضح أدناه.



المصدر:

https://heartbeat.comet.ml/colorizing-b-w-images-with-gans-in-tensorflow-f444f737db6c

Generative Adversarial Networks

Deep Learning Projects Solved with Generative Adversarial Networks (GANs)

By: Dr. Alaa Taima

GANs

